UNIVERSIDADE FEDERAL DO RIO GRANDE
CENTRO DE CIÊNCIAS COMPUTACIONAIS
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO
CURSO DE MESTRADO EM ENGENHARIA DE COMPUTAÇÃO

Dissertação de Mestrado

Testes em Sistemas Multiagentes sob Modelo Organizacional $\mathcal{M}$oise^{+}

Ricardo Arend Machado

Dissertação de Mestrado apresentado ao Programa de Pós-Graduação em Computação da Universidade Federal do Rio Grande, como requisito parcial para a obtenção do grau de Mestre em Engenharia de Computação

Orientador: Prof. Dr. Eder Mateus Nunes Gonçalves

Rio Grande, 2020

Ficha Catalográfica

M149t Machado, Ricardo Arend.
Testes em sistemas multiagentes sob modelo organizacional Moise+ / Ricardo Arend Machado. – 2020.
78 f.

Dissertação (mestrado) – Universidade Federal do Rio Grande – FURG, Programa de Pós-Graduação em Computação, Rio Grande/RS, 2020.
Orientador: Dr. Eder Mateus Nunes Gonçalves.

1. Testabilidade 2. Moise 3. Sistemas Multiagentes 4. Rede de Petri 5. Organização I. Gonçalves, Eder Mateus Nunes II. Título.

CDU 004.4

Catalogação na Fonte: Bibliotecário José Paulo dos Santos CRB 10/2344

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO RIO GRANDE
CENTRO DE CIÊNCIAS COMPUTACIONAIS
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO
CURSO DE MESTRADO EM ENGENHARIA DE COMPUTAÇÃO

DISSERTAÇÃO DE MESTRADO

Testes em Sistemas Multiagentes sob Modelo Organizacional Moise+

Ricardo Arend Machado

Banca examinadora:

Prof. Dr. Rafael Heitor Bordini – PUC-RS

Prof. Dr. Alison Roberto Panisson - UFPEL

Prof. Dr. Diana Adamatti - FURG

Prof. Dr. Eder Mateus Nunes Gonçalves
Orientador(a)

Dedico a conclusão desse trabalho à minha mãe Erselha e minha namorada Kennah pela paciência e apoio.

AGRADECIMENTOS

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

RESUMO

MACHADO, Ricardo Arend. **Testes em Sistemas Multiagentes sob Modelo Organizacional $\mathcal{M}$oise$^+$**. 2020. 78 f. Dissertação (Mestrado) – Programa de Pós-Graduação em Computação. Universidade Federal do Rio Grande, Rio Grande.

A fase de teste é uma etapa crucial para buscar a correção de todo sistema de software de modo a dar garantias de funcionamento e segurança para os usuários. Porém o teste em Sistemas Multiagentes (SMA) é uma tarefa desafiadora devido ao comportamento autônomo, proativo e não-determinístico dos agentes, o que faz com que seja muito difícil prever todas as possibilidades de cenários necessários para sua completa validação.

Quanto a dimensão social na concepção de SMA's, modelos organizacionais impõem restrições à atuação dos agentes que o constituem, coordenando ações e estabelecendo regras de comportamento. Apesar do nível maior de controle sobre os agentes, o sistema não fica livre de comportamentos imprevisíveis que fogem ao controle do projetista. Assim técnicas de teste voltadas especificamente para SMA se justificam para aumentar a confiabilidade da aplicação.

Nesse trabalho é apresentado um método para o dimensionamento e sistemática de testes em SMA especificados sob o modelo organizacional $\mathcal{M}$oise$^+$. Para isso, propõe-se um mapeamento em uma Rede de Petri Colorida que dimensiona o número de testes necessários à validação de uma especificação $\mathcal{M}$oise$^+$, e uma sistemática para a geração dos casos de testes. A validação deu-se através de testes de sistema utilizando casos de testes gerados a partir de exemplos clássicos na literatura sobre $\mathcal{M}$oise$^+$. O resultado é uma metodologia de testes para o nível social de SMA's especificados através deste modelo organizacional.

Palavras-chave: Testabilidade, Moise, Sistemas Multiagentes, Rede de Petri, Organização.

ABSTRACT

MACHADO, Ricardo Arend. **Tests on Multiagent Systems under Organizational Model $\mathcal{M}$oise$^+$**. 2020. 78 f. Dissertação (Mestrado) – Programa de Pós-Graduação em Computação. Universidade Federal do Rio Grande, Rio Grande.

The test phase is a crucial step to seek the correction of the entire software system in order to provide guarantees of operation and safety for users. However, testing in Multi-Agent Systems (MAS) is a challenging task due to the autonomous, proactive and non-deterministic behavior of agents, which makes it very difficult to predict all the possible scenarios necessary for its complete validation.

As for the social dimension in the conception of MAS, organizational models impose restrictions on the performance of the agents that constitute it, coordinating actions and establishing rules of behavior. Despite the greater level of control over agents, the system is not free from unpredictable behaviors that are beyond the control of the designer. Thus, testing techniques aimed specifically at SMA are justified to increase the reliability of the application.

This work presents a method for the dimensioning and systematic testing of MAS specified under the $\mathcal{M}$oise$^+$ organizational model. For this, is proposed a mapping in a Coloured Petri Net that scales the number of tests necessary to validate a $\mathcal{M}$oise$^+$ specification, and a systematic way to generate test cases. The validation took place through system tests using test cases generated from classic examples in the literature about $\mathcal{M}$oise$^+$. The result is a testing methodology for the social level of MAS.

Keywords: Testability, Multi-Agent Systems, Petri Net, Moise, Organization.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SMA	Sistema Multiagente
EE	Especificação Estrutural
EF	Especificação Funcional
ED	Especificação Deôntica
ES	Esquema Social
RP	Rede de Petri
RPC	Rede de Petri Colorida
BDI	*Belief Desire Intentions*

SUMÁRIO

1 INTRODUÇÃO

Os Sistemas Multiagente (SMA) ganham cada vez maior importância nas mais diferentes áreas devido às características únicas dos agentes, como reatividade, proatividade, autonomia e capacidade social (PADGHAM; WINIKOFF, 2005). Porém, certas aplicações exigem um mínimo de confiabilidade para que o sistema não apresente nenhum risco ao usuário. Podemos citar como exemplo o controle de veículos autônomos, aplicações militares, gerenciamento de sistemas de distribuição de energia elétrica (*Smart Grids*), logística, etc.

Para conseguir essa garantia testes de software são necessários. O teste de software consiste na verificação dinâmica do comportamento de um programa em um conjunto de casos de teste selecionados adequadamente (ABRAN et al., 2004). Vale destacar que o teste de software é uma atividade importante que abrange todo o processo de desenvolvimento e manutenção (ADRION; BRANSTAD; CHERNIAVSKY, 1982). O problema é que testar um SMA não é uma tarefa simples pois esses sistemas costumam ser programados para serem autônomos e deliberativos, e operam em um mundo aberto, o que os tornam sensíveis ao contexto (HOUHAMDI, 2011).

Uma das maneiras de restringir esse comportamento dos SMA é representá-los como um grupo através de modelos organizacionais. Tais modelos coordenam os agentes em grupos e hierarquias além de estabelecer regras comportamentais específicas (ARGENTE; JULIAN; BOTTI, 2006). Um desses modelos, chamado $\mathcal{M}$oise$^+$ (HÜBNER; SICHMAN; BOISSIER, 2002; HÜBNER, 2003; HUBNER; SICHMAN; BOISSIER, 2004), possui: uma *especificação estrutural* que define a quais grupos pertencem os papéis, a relação existente entre os diferentes papéis e a quantidade máxima de agentes que podem desempenhar um determinado papel; uma *especificação funcional* que define a ordem que as metas serão realizadas e relaciona as metas com as missões. Essas especificações ficaram mais independentes que no trabalho anterior, precisando assim de uma *especificação deôntica* responsável pela ligação entre elas.

Como exemplos de trabalhos recentes relacionados com o tema podemos citar Rehman e Nadeem (2015) que apresenta uma abordagem para teste em SMA baseada em modelos utilizando a ferramenta Prometheus. Também utilizando modelos temos Kerra-

oui et al. (2016) que propôs uma abordagem utilizando redes de Petri para modelagem. Já Winikoff (2017) traz uma análise da testabilidade de sistemas BDI. Barnier et al. (2017) apresenta uma nova estratégia para testes em sistemas multiagentes embarcados. Embora esses trabalhos trouxeram uma contribuição significativa para a área de testes em SMA, nenhum deles levou em consideração o impacto que as restrições de uma camada organizacional pode gerar para os testes do sistema.

Como visto nos trabalhos acima citados, testes de sistemas baseados em modelos são bastantes utilizados e a rede de Petri é uma ferramenta de modelagem gráfica e matemática aplicável a muitos desses sistemas. A rede de Petri é um modelo formal e pode ser usada como um auxílio de comunicação visual similar a fluxogramas, diagramas de blocos e redes.

Mesmo um SMA tendo sua concepção baseada em uma organização e, possuindo uma serie de restrições para ações dos agentes, não se pode dar uma garantia de funcionamento sem realizar testes. Nesse sentido, o objetivo desta dissertação é propor uma metodologia para testes de especificações $\mathcal{M}$oise$^+$ constituída por: (i) um método formal para avaliação da testabilidade do modelo em análise, de modo a dimensionar o número de casos de testes necessários a sua validação; (ii) uma sistemática para a geração dos casos de teste, ou seja, os cenários que visam a cobertura de todas as possibilidades de execução do modelo. A testabilidade é avaliada por uma extensão a Redes de Petri Colorida que incorpora elementos de uma especificação $\mathcal{M}$oise$^+$. O resultado esperado da metodologia é um conjunto de ferramentas que medem a confiabilidade da especificação $\mathcal{M}$oise$^+$ gerada.

O trabalho está organizado da seguinte maneira: no capítulo 2 temos um referencial teórico abordando os temas agentes e SMA, organizações, $\mathcal{M}$oise$^+$, teste de software, teste de SMA, redes de Petri, redes de Petri coloridas e a ferramenta CPN Tools. Já no capítulo 3 alguns trabalhos relacionados com o objetivos e temas semelhantes são apresentados. No capítulo 4 a proposta do trabalho é apresentada com a metodologia utilizada. No capítulo 5 dois exemplos de caso de uso são apresentados. O capítulo 6 é a conclusão.

2 REFERENCIAL TEÓRICO

Nesse capítulo é apresentado um referencial teórico a respeito dos temas que fazem parte do desenvolvimento desse trabalho. O conteúdo desse referencial está dividido da seguinte maneira: na primeira seção 2.1 temos uma visão sobre agentes e SMA para entender melhor o funcionamento e as características específicas desses sistemas além de uma descrição sobre organizações para SMA. Na seção 2.2 a organização $\mathcal{M}$oise$^+$ que é utilizada no trabalho será apresentada. As seções 2.3 e 2.4 mostram o teste de software e o teste de SMA respectivamente, para apresentar seus tipos, características de cada um e definir melhor essa área e sua importância no desenvolvimento do sistema. A seguir nas seções 2.5 e 2.6 as redes de Petri ordinárias e coloridas são definidas mostrando sua estrutura, mecanismos de funcionamento e as características para cada tipo. Por fim a seção 2.7 apresenta a ferramenta CPN Tools que é utilizada no trabalho para a criação das redes de Petri coloridas.

2.1 Agentes e Sistemas Multiagentes

2.1.1 Agentes

Embora existam diferentes definições para o termo agente, apenas duas delas serão aqui apresentadas por se tratarem de definições de autores bastante conhecidos na área. A primeira foi proposta por Russell, Norvig et al. (2003) que diz:

> "Um agente é qualquer coisa que possa perceber seu ambiente através de sensores e atuar sobre esse ambiente através de atuadores."

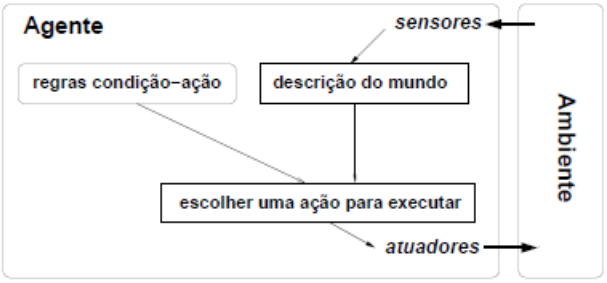
Outra definição mais detalhada foi proposta por Ferber e Gasser (1991) onde:

> "Um agente é uma entidade real, ou virtual, imersa em um dado ambiente onde está habilitado a executar algumas ações, perceber e representar parcialmente este ambiente, podendo ainda comunicar-se com os demais agentes do ambiente. Este agente apresenta um comportamento autônomo que é uma consequência de suas observações, do conhecimento armazenado e das interações com os demais agentes do ambiente"

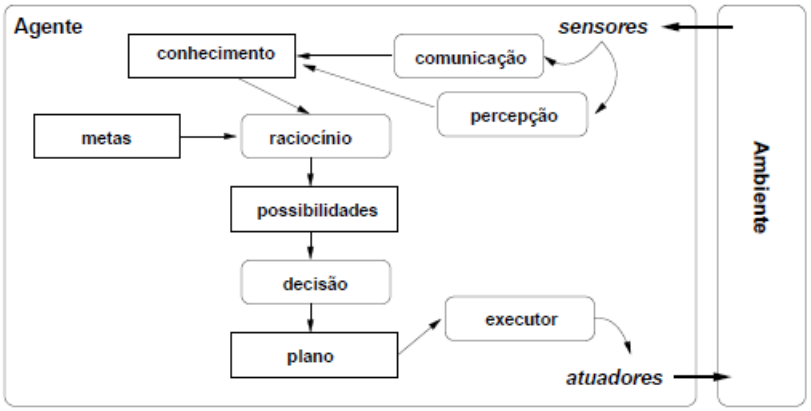
Assim agentes são entidades autônomas, que gerenciam por si mesmos seu próprio estado e comportamento. Os agentes podem se comunicar com outros agentes ou usuários além de interagir com o ambiente, por meio de protocolos de comunicação e interação.

Os agentes são diferentes de outros programas de software devido às suas características, sendo elas (WOOLDRIDGE; JENNINGS, 1995) :

- **Autonomia:** os agentes são independentes e tomam suas próprias decisões, sem intervenções diretas de outros agentes ou seres humanos.
- **Reatividade:** os agentes reagem às mudanças que ocorrem no ambiente em que estão situados.
- **Pró-atividade:** os agentes agem de maneira direcionada aos objetivos.
- **Sociabilidade:** os agentes interagem com outros agentes através de algum tipo de protocolo de comunicação.



(a) Agente Reativo



(b) Agente Cognitivo

Figura 1: Duas arquiteturas clássicas de agentes: (a) Agente Reativo (RUSSELL; NORVIG, 1995), (b) Agente Cognitivo (DEMAZEAU; MÜLLER, 1990)

Existem dois modelos de funcionamento interno dos agentes sendo eles: agentes reativos e agentes cognitivos. Os agentes reativos, Figura 1(a), têm funcionamento simples pois escolhem suas ações baseados unicamente nas percepções que têm do ambiente. Já os agentes cognitivos, Figura 1(b), são mais complexos pois possuem um estado mental e funcionam de forma a raciocinar para construir um plano de ações que leva a realização do objetivo pretendido.

Quanto a arquitetura dos agentes, que modela seu comportamento, podemos destacar a arquitetura BDI (*Belief Desire Intention*) (RAO; GEORGEFF, 1991), baseada num modelo de comportamento humano desenvolvido por um filósofo (BRATMAN, 1987). Existem atualmente muitas plataformas de implementação de agentes baseadas no modelo BDI como JAM (HUBER, 1999), JACK (WINIKOFF, 2005), JADEX (BRAUBACH; LAMERSDORF; POKAHR, 2003) e o JASON (BORDINI; HÜBNER; WOOLDRIDGE, 2007). Agentes BDI trabalham com a ideia de crenças, desejos e intenções e na Figura 2 podemos ver um diagrama representando a arquitetura BDI (WOOLDRIDGE, 1999) contendo sete componentes principais onde:

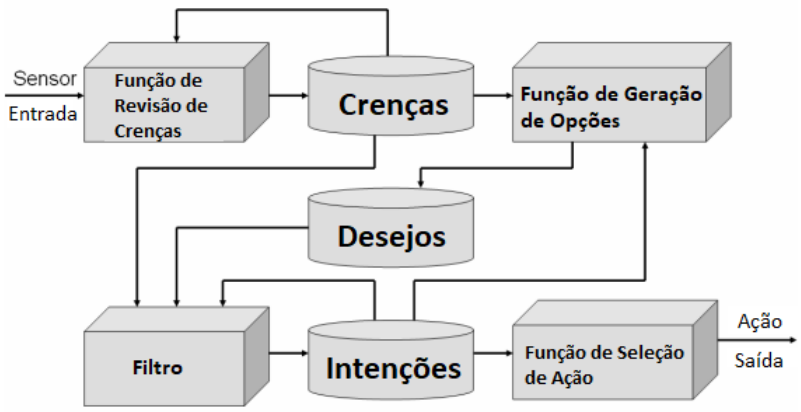


Figura 2: Arquitetura BDI genérica (WOOLDRIDGE, 1999)

- **Crenças** (do inglês *Beliefs*): um conjunto de crenças atuais representando a informação que o agente tem sobre seu ambiente;

- **Função de Revisão de Crenças**: determina um novo conjunto de crenças a partir da percepção da entrada e das crenças do agente;

- **Função de Geração de Opções**: determina as opções disponíveis ao agentes (seus desejos), com base nas suas crenças sobre seu ambiente e nas suas intenções;

- **Desejos** (do inglês *Desires*): um conjunto de estados ou objetivos que um agente deseja alcançar;

- **Filtro**: uma função de filtro, a qual representa o processo de deliberação do agente, que determina as intenções do agente com base nas suas crenças, desejos e intenções atuais;
- **Intenções** (do inglês *Intentions*): um conjunto de intenções atual, que representa o foco atual do agente, isto é, aqueles estados que o agente está determinado a alcançar;
- **Função de Seleção de Ação**: determina uma ação a ser executada com base nas suas intenções atuais.

2.1.2 Sistemas Multiagentes

Um sistema que consiste em um grupo de agentes que potencialmente podem interagir entre si é chamado de sistema multiagente (SMA). Mais especificamente um SMA é uma sociedade de agentes autônomos, que evolui em cooperação em seu ambiente para atingir coletivamente um objetivo global (BARNIER et al., 2017).

Os SMA possuem características que o diferem de outros sistemas, a saber (VLASSIS, 2007):

- **Design do agente:** Frequentemente os agentes de um mesmo SMA são projetados de maneiras diferentes como, por exemplo, na parte comportamental. A heterogeneidade do agente pode afetar todos os aspectos funcionais de um agente, desde a percepção até a tomada de decisão.
- **Ambiente:** Os agentes precisam lidar com ambientes que podem ser estáticos ou dinâmicos. Em um SMA, a mera presença de múltiplos agentes faz o ambiente parecer dinâmico do ponto de vista de cada agente.
- **Percepção:** O fato de os agentes poderem observar coisas diferentes torna o mundo parcialmente observável para cada agente, o que tem várias consequências na tomada de decisão dos agentes.
- **Controle:** Ao contrário dos sistemas de agente único, o controle em um SMA é tipicamente descentralizado. O controle descentralizado é preferível ao controle centralizado por razões de robustez e tolerância a falhas.
- **Conhecimento:** Em um SMA, os níveis de conhecimento de cada agente sobre o estado atual do mundo podem diferir substancialmente. Assim cada agente deve também considerar o conhecimento de cada outro agente em sua tomada de decisão.
- **Comunicação:** A interação é frequentemente associada a alguma forma de comunicação. A comunicação pode ser usada em vários casos, por exemplo, para coordenação entre agentes cooperativos ou para negociação entre agentes de interesse próprio.

A classificação de um SMA é uma questão complexa, pois pode ser feita com base em vários atributos diferentes como arquitetura, aprendizagem, comunicação e coordenação. Considerando apenas a arquitetura interna dos agentes que formam um SMA, esse sistema pode ser classificado em dois tipos (GOKULAN; SRINIVASAN, 2010) :

- **Estrutura Homogênea:** Em uma arquitetura homogênea, todos os agentes que formam o sistema multiagente têm a mesma arquitetura interna. A arquitetura interna refere-se aos objetivos locais, capacidades do sensor, estados internos, mecanismos de inferência e ações possíveis.

- **Estrutura Heterogênea:** Em uma arquitetura heterogênea, os agentes podem diferir em capacidade, estrutura e funcionalidade. A arquitetura heterogênea ajuda a tornar os aplicativos de modelagem muito mais próximos do mundo real.

A comunicação é um dos componentes cruciais em sistemas multiagentes e pode ser classificada em dois tipos: comunicação local (*peer to peer*) ou quadro negro (*blackboard*). Na comunicação local os agentes trocam mensagens entre si diretamente sem a necessidade de um local para armazenamento da informação, esse tipo de comunicação é indicada para aplicações descentralizadas. Já na comunicação por quadro negro um grupo de agentes compartilha um mesmo repositório de dados para enviar e receber mensagens (GOKULAN; SRINIVASAN, 2010).

2.1.3 Organizações

A organização de SMA pode ser entendida a partir de duas perspectivas: organização como um processo e organização como uma entidade. Ou seja, a organização é considerada tanto como o processo de organização de um conjunto de indivíduos, ou como uma entidade em si, com seus próprios requisitos e objetivos (DIGNUM, 2009).

Existem diversos conceitos e definições sobre organização, uma dessas definições, dada por (WOOLDRIDGE; JENNINGS; KINNY, 2000) diz:

> "Vemos uma organização como uma coleção de papéis, que estão em certos relacionamentos uns com os outros, e que participam de padrões sistemáticos institucionalizados de interações com outros papéis."

Alguns conceitos importantes e que são recorrentes quando se fala de organização em SMA serão listados a seguir:

- **Papel:** é uma descrição de um comportamento abstrato dos agentes que descreve as restrições que um agente terá que satisfazer para obter uma função, os benefícios que um agente receberá ao desempenhar essa função e as responsabilidades associadas a essa função (FERBER; GUTKNECHT; MICHEL, 2003).

- **Meta:** pode ser meta global, que é o estado do mundo desejado pelo SMA ou meta local, que é o objetivo de um único agente (HÜBNER, 2003).
- **Plano:** um conjunto de ações que o agente executa para cumprir uma determinada meta.

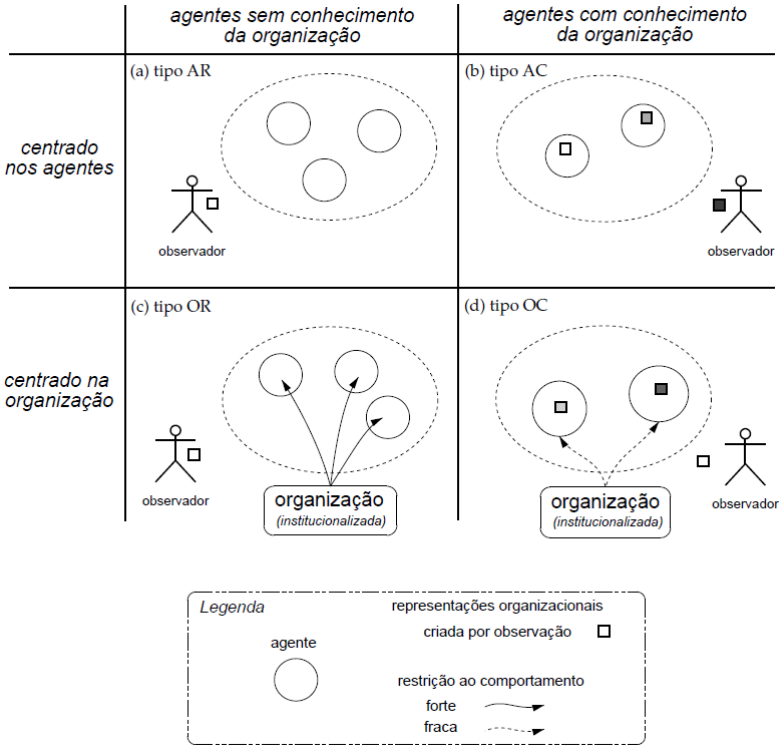


Figura 3: Tipos de Organização (HÜBNER, 2003)

Hübner (2003) classificou as organizações de SMA em quatro tipos diferentes: AR, AC, OR, OC. Essa classificação está ilustrada na Figura 3, onde:

- **Organização tipo AR:** possui uma visão centrada nos agentes que não representam nem raciocinam sobre a organização observada.
- **Organização tipo AC:** possui uma visão centrada nos agentes, porém esses agentes tem a capacidade de representar e raciocinar sobre a organização observada.
- **Organização tipo OR:** possui uma visão centrada na organização e os agentes não tem a capacidade de gerar uma representação dela o que implica que esses agentes não raciocinam sobre a organização e nem tiram proveito do conhecimento para realizar suas tarefas. O nível de restrição forte implica que os agentes não podem se comportar de maneira diferente daquela estabelecida pela organização.
- **Organização tipo OC:** possui uma visão centrada na organização e os agentes têm a capacidade de gerar uma representação interna sobre ela. Isso implica que os

agentes tem seu comportamento parcialmente determinado pela organização pois eles raciocinam sobre ela e utilizam essas informações para melhorar seu funcionamento. A restrição fraca indica que os agentes podem não seguir o comportamento estabelecido pela organização.

No começo, o desenvolvimento de SMA era focado no comportamento individual do agente, sendo esse método conhecido como abordagem centrada em agentes. Porém, se tornou necessário a criação de uma novo método onde a unidade conceitual básica é a organização multiagente, como um todo, e não o nível dos agentes individuais. Para isso a orientação deixou de ser o comportamento do agente e passou a ser a organização sendo esse método conhecido como abordagem centrada na organização (LEMAÎTRE; EXCELENTE; FALLAH-SEGHROUCHNI, 1999).

Com o passar do tempo, diferentes modelos organizacionais surgiram. Esses modelos fazem uma descrição explícita de um SMA determinando por exemplo o que constitui a organização (HÜBNER, 2003). Um desses modelos que iremos apresentar na próxima seção é chamado de $\mathcal{M}$oise$^+$.

2.2 $\mathcal{M}$oise$^+$

O modelo organizacional $\mathcal{M}$oise foi escrito inicialmente em (HANNOUN et al., 2000). Esse modelo foi estruturado em três níveis:

- **Individual:** Os comportamentos pelos quais um agente é responsável quando adota uma função.
- **Social:** As interconexões entre os papéis.
- **Coletivo:** Estruturação global e interconexão dos agentes e das estruturas entre si.

A principal deficiência do $\mathcal{M}$oise é a falta do conceito de um plano global explícito no modelo e a forte dependência entre a estrutura e o funcionamento (HÜBNER; SICHMAN; BOISSIER, 2002).

Então, o modelo foi aprimorado e uma nova extensão foi desenvolvida chamada de $\mathcal{M}$oise$^+$ que teve como principal objetivo corrigir a deficiência de seu antecessor. O modelo $\mathcal{M}$oise$^+$ serve para representar tanto uma organização institucionalizada (tipo OR e OC) quanto a observada (tipo AR e AC) (HÜBNER, 2003). A especificação organizacional desse modelo contém uma especificação estrutural (EE) e uma especificação funcional (EF) de forma independente e unidos pela especificação deôntica (ED) (HÜBNER; SICHMAN; BOISSIER, 2002).

2.2.1 Especificação Estrutural

A estrutura do modelo $\mathcal{M}$oise$^+$ também possui os níveis individual, social e coletivo da versão anterior, porém as descrições desses níveis são um pouco diferentes. No nível individual o papel tem a função de ser o elo entre o agente e a organização. No nível social os papéis estão relacionados a outros papéis representando restrições impostas às interações de um papel com outros. No nível coletivo o grupo representa um conjunto de agentes com afinidades maiores e objetivos mais próximos (HÜBNER, 2003).

Na Figura 4 é possível visualizar um exemplo de EE para um modelo chamado *Writing Paper* (HÜBNER; BOISSIER; BORDINI, 2011). Nela podemos ver o grupo (*wpgroup*), os papéis (*escritor, editor e autor*) e as ligações entre eles.

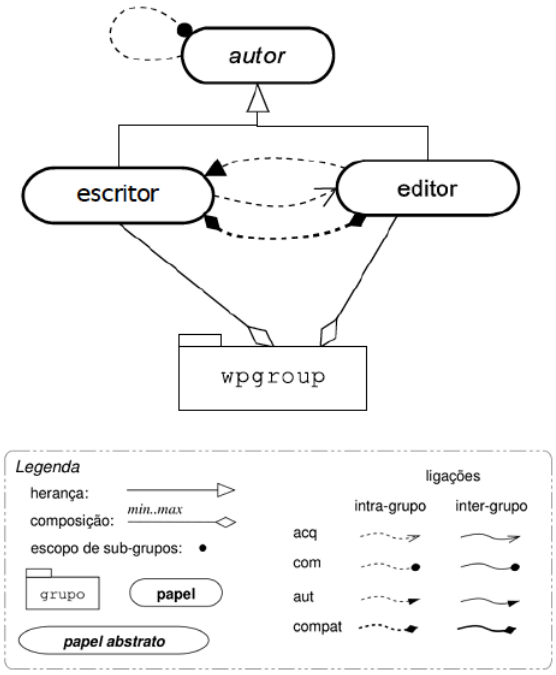


Figura 4: Exemplo de Especificação Estrutural (HÜBNER; BOISSIER; BORDINI, 2011).

2.2.2 Especificação Funcional

A especificação funcional no $\mathcal{M}$oise$^+$ é constituída por um conjunto de esquemas sociais (ES) e uma relação de preferência de missões. Para os esquemas sociais é utilizado o conceito de meta global. Uma meta global representa um estado do mundo desejado pelo SMA, com outro contexto da meta local que seria o objetivo de um único agente. Essa EF possui dois níveis sendo o nível individual onde o ES é constituído por missões e o nível coletivo onde o próprio ES é uma árvore de decomposição de metas globais onde a raiz é a meta do ES e a decomposição das metas é feita através de planos (HÜBNER, 2003).

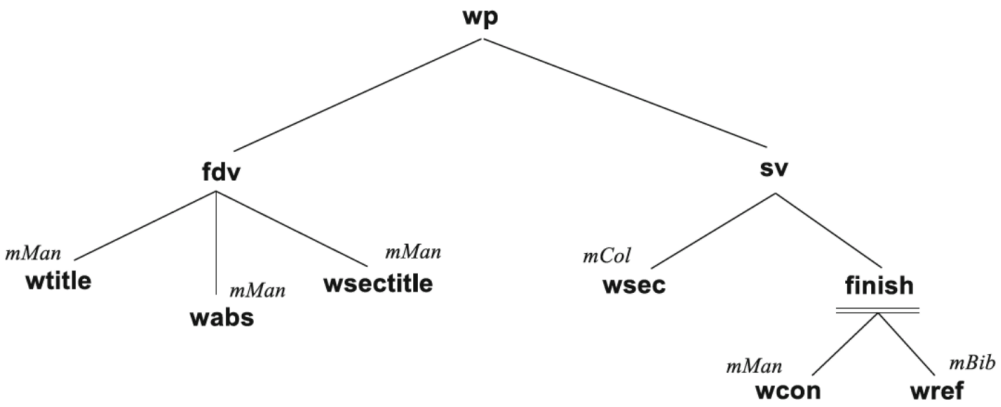


Figura 5: Exemplo de Especificação Funcional (HÜBNER; BOISSIER; BORDINI, 2011).

A Figura 5 apresenta uma árvore de decomposição de metas de um ES para o exemplo *Writing Paper* onde o objetivo é escrever um artigo. A leitura deve ser realizada da esquerda para a direita e de baixo para cima. Por exemplo, neste esquema, primeiro um agente que assume a missão *mMan* deve escrever uma versão de rascunho do artigo (*fdv*), contendo como sub-metas escrever um título (*wtitle*), um resumo (*wabs*) e o título das seções (*wsectitle*), sendo necessário atingir os objetivos nesta sequência.

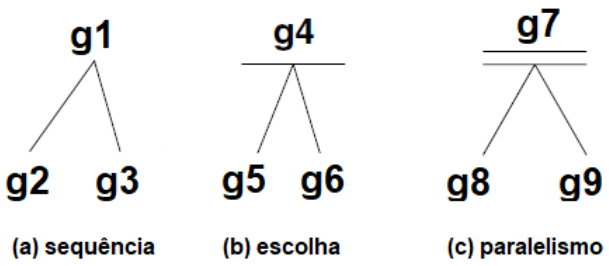


Figura 6: Tipos de Operadores de Plano.

No $\mathcal{M}$oise$^+$ existem três tipo de operadores usados para a construção de um plano como mostra a Figura 6:

- **Sequência** "$,$"**:** No plano "$g1 = g2, g3$" a meta $g1$ será satisfeita se a meta $g2$ e $g3$ forem satisfeitas nessa ordem;
- **Escolha** "$|$"**:** No plano "$g4 = g5|g6$" será satisfeita a meta $g4$ caso uma das metas $g5$ ou $g6$ for satisfeita;
- **Paralelismo** "$||$"**:** No plano "$g7 = g8||g9$" será satisfeita a meta $g7$ se ambas $g8$ e $g9$ forem satisfeitas paralelamente.

2.2.3 Especificação Deôntica

Na especificação deôntica são definidos quais missões um papel possui permissão ou obrigação de realizar, assim a ED é uma relação entre a EF e a EE. Uma permissão determina que um agente com o papel *p* pode se comprometer com a missão *m*. Já uma obrigação estabelece que um agente com o papel *p* é obrigado a se comprometer com a missão *m* (HÜBNER, 2003). Por exemplo, no caso do *Writing Paper*, onde a ED pode ser vista na Tabela 1, podemos dizer que o papel *editor* tem a permisão de realizar a missão $mMan$, assim sendo qualquer agente com o papel *editor* poderá realizar uma meta que tenha relação com $mMan$.

papel	relação deôntica	missão
editor	permissão	$mMan$
writer	obrigação	$mCol$
writer	obrigação	$mBib$

Tabela 1: Exemplo de Especificação Deôntica, adaptado de (HÜBNER; BOISSIER; BORDINI, 2011).

2.3 Teste de Software

O teste é uma fase de desenvolvimento de software com o objetivo de avaliar a qualidade do produto e aprimorá-lo, detectando erros e problemas. Embora existam diferentes opiniões sobre uma definição formal do termo testar, podemos citar uma, proposta em (IEEE..., 1994), que é bastante utilizada:

> "Testar é o processo de análise de um item de software para detectar a diferença entre condições existentes e necessárias (ou seja, *bugs*) e avaliar os recursos do item de software."

O teste preocupa-se em estabelecer que um sistema desenvolvido funcione adequadamente, levando em consideração os requisitos do usuário e a especificação do sistema associado. Estes dois aspectos essenciais dos testes são comumente referidos como validação e verificação (ABDULLAH; BURNSTEIN, 2003). Definições para esses termos podem ser encontradas em (IEEE..., 1990) sendo:

> "**Validação** é processo de avaliar um sistema ou componente durante ou no final do processo de desenvolvimento para determinar se ele satisfaz os requisitos especificados."

> "**Verificação** é o processo de avaliar um sistema ou componente para determinar se os produtos de uma determinada fase de desenvolvimento satisfazem as condições impostas no início dessa fase."

2.3.1 Cobertura de Teste

Na prática de teste de software, os testadores geralmente precisam gerar casos de teste para executar todas as instruções no programa pelo menos uma vez. Um caso de teste é uma entrada na qual o programa em teste é executado durante o teste. Um conjunto de testes é um conjunto de casos de teste para testar um programa (ZHU; HALL; MAY, 1997).

Para alcançar uma boa cobertura de teste é necessária a escolha de um critério de adequação de teste apropriado para o sistema a ser testado. Formalmente, um critério de adequação pode ser definido como um predicado que afirma se um conjunto de casos de teste é adequado para testar um programa em relação a uma determinada especificação (ZHU, 1996). Abaixo podemos ver alguns exemplos de critérios de adequação:

- **Instruções (*statement*):** O critério de adequação por instruções é atendido por um conjunto de testes específico para um programa específico se cada instrução executável no programa for executada por pelo menos um caso de teste no conjunto de testes.

- **Ramificações (*branch*):** Esse critério mede um programa pelo número de ramificações que ele contém e informa aos usuários quais dessas ramificações não foram obtidas durante a execução do teste.

- **Caminhos (*path*):** O critério todos os caminhos exige que todos os caminhos de execução desde a entrada do programa até sua saída sejam executados durante o teste.

Esses foram alguns exemplos de critérios de adequação, porém diversos critérios de adequação de testes foram propostos na literatura (ZHU; HE, 2002; AMMANN; OFFUTT, 2016; MATHUR, 2013) como transição, mutação, fluxo de dados, etc.

2.3.2 Níveis de Teste

O teste pode ser realizado em várias etapas do desenvolvimento de um software. Essas etapas podem ser vistas como níveis de recursos que precisam ser testados. (IEEE..., 1994) descreve esses níveis como:

- **Teste de componente:** testes realizados para verificar a implementação do projeto para um elemento de software ou uma coleção de elementos de software.

- **Teste de integração:** Uma progressão ordenada de testes em que elementos de software, elementos de hardware ou ambos são combinados e testados até que todo o sistema tenha sido integrado.

- **Teste de sistema:** é o processo de testar um sistema integrado de hardware e um sistema de software para verificar se o sistema atende aos requisitos especificados.

- **Teste de aceitação:** testes formais realizados para determinar se um sistema satisfaz ou não seus critérios de aceitação e para permitir que o cliente determine se aceita ou não o sistema.

Um procedimento importante durante o teste de software é a geração de casos de teste. Um caso de teste especifica valores reais de entrada e resultados esperados. O objetivo na geração de casos de teste é exercitar a lógica do componente e configurar cenários de teste que exponham falhas, omissões e resultados inesperados (IEEE. . . , 1994).

No caso de SMA, existem algumas características que torna o teste mais complexo do que em sistemas convencionais. Na próxima seção é apresentada uma descrição desse assunto com maior profundidade.

2.4 Teste de Sistemas Multiagentes

Uma das grandes dificuldades ao trabalhar com SMA está relacionado com a realização de testes. No que diz respeito aos sistemas multiagentes, o teste é uma tarefa desafiadora, e requer novas técnicas que lidem com sua natureza específica. Segundo (HOUHAMDI, 2011) as principais razões para esse grau de dificuldade são:

- **Maior complexidade:** existem vários processos distribuídos que são executados de forma autônoma e simultânea;

- **Quantidade de dados:** os sistemas podem ser compostos por milhares de agentes, cada um com seus próprios dados;

- **Não-determinismo:** não há garantia que duas execuções do mesmo sistema com as mesmas entradas levem ao mesmo estado, dificultando a procura de erros;

- **Alcançabilidade:** uma vez que não é possível determinar antecipadamente todas as interações de um agente durante sua execução.

Assim como no teste de software, nos SMA existem diferentes níveis de testes que podem ser aplicados. Uma classificação foi descrita por (NGUYEN, 2009) que separa os testes em: unidade, agente, integração, sistema e aceitação. A descrição individual desses níveis pode ser vista a seguir:

- **Unidade:** Testar unidades de código e módulos que compõem os agentes como metas, planos, crenças, sensores, mecanismo de raciocínio e assim por diante.

- **Agente:** Testar a integração dos diferentes módulos dentro de um agente; testar os recursos dos agentes para preencher seus objetivos e detectar o ambiente.

- **Integração:** Testar a interação de agentes, protocolos de comunicação e semântica, interação de agentes com o ambiente, integração de agentes com recursos compartilhados, cumprimento de normas; observar propriedades emergentes; certificar que um grupo de agentes e os recursos do ambiente funcionem corretamente juntos.

- **Sistema:** Testar o SMA como um sistema em execução no ambiente operacional de destino; teste para propriedades de qualidade que o sistema pretendido deve atingir, como adaptação, abertura, tolerância a falhas, desempenho.

- **Aceitação:** Testar o SMA no ambiente de execução do cliente e verificar se ele atende aos objetivos das partes interessadas.

Embora existam diferentes níveis de teste as dificuldades para se testar um SMA fazem com que existam poucas ferramentas disponíveis para testes, no capítulo 3 são apresentadas algumas dessas ferramentas. A criação de novas metodologias é necessária para que essa área evolua e consiga tornar os SMA competitivos no mercado atual.

2.5 Redes de Petri

Rede de Petri (RP) é uma ferramenta de modelagem gráfica e matemática aplicável a muitos sistemas. Elas pode ser usada como um auxílio de comunicação visual similar a fluxogramas, diagramas de blocos e redes. Na figura 7 podemos visualizar um exemplo de uma RP, que possuem os seguintes elementos básicos (CARDOSO; VALETTE, 1997):

- **Lugar:** representado por um círculo, que pode ser interpretado como uma condição, um estado parcial, uma espera, um procedimento, etc. Em geral, todo lugar tem um predicado associado. No exemplo da Figura 7 temos *máquina livre*, *peça em espera* e *máquina em operação*.

- **Transição:** representada por barra ou retângulo, é associada a um evento que ocorre no sistema, como o evento iniciar a operação representado pela transição *t*.

- **Ficha:** representado por um ponto num lugar, é um indicador significando que a condição associada ao lugar é verificada. Pode representar um objeto ou uma estrutura de dados que se manipula. Por exemplo, uma ficha no lugar máquina livre indica que a máquina está livre. Se não tem fichas neste lugar indica que a máquina não está livre, estando assim em operação (Figura 7b). Se no lugar peças em espera houvesse três fichas, indicaria que existem três peças em espera.

Embora a representação gráfica seja uma vantagem da RP, a característica mais importante deste modelo é o fato de ser formal. Portanto, formalmente uma RP é descrita por (MURATA, 1989) como:

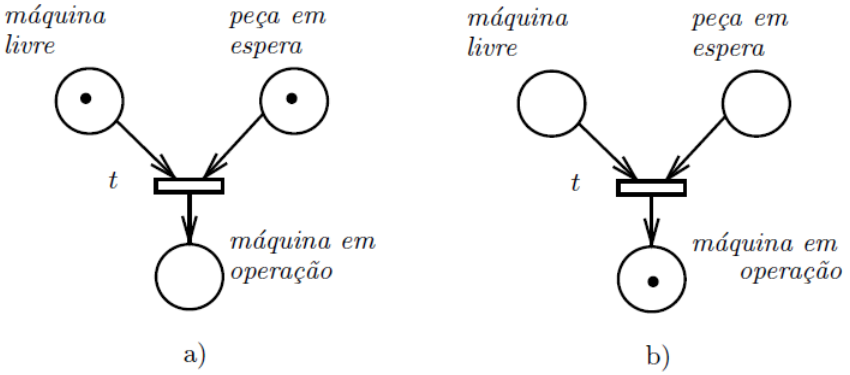


Figura 7: Rede de Petri (CARDOSO; VALETTE, 1997)

Definição 1. ***(Rede de Petri)*** *Uma rede de Petri é uma 5-tupla* $RP = (P, T, F, W, M_o)$ *onde:*

1. $P = \{p1, p2, ..., p_m\}$ *é um conjunto finito de lugares,*
2. $T = \{t1, t2, ..., t_n\}$ *é um conjunto finito de transições,*
3. $F \subseteq (P \times T) \cap (T \times P)$ *é um conjunto de arcos (relação de fluxo),*
4. $W : F \rightarrow \{0, 1, 2, ...\}$ *é uma função de peso,*
5. $M_0 : P \rightarrow \{0, 1, 2, ...\}$ *é a marca inicial.*

Nas RP ordinárias existe apenas um tipo de marca, o que não permite a diferenciação de recursos em um lugar, sendo necessários lugares distintos para expressarmos recursos similares. Assim, essas redes se mostraram inadequadas para a aplicação de sistemas reais já que os modelos obtidos possuem grandes dimensões além de serem não-hierárquicos dificultando sua compreensão. Isso motivou a criação de extensões como a rede de Petri Colorida (RPC) que é apresentada na próxima seção (MACIEL; LINS; CUNHA, 1996).

Existem algumas propriedades importantes relativas às redes de Petri, como vivacidade, reiniciabilidade e limitabilidade, e suas definições implicam considerações sobre o conjunto de marcações acessíveis a partir da marcação inicial. Essas propriedades estão abaixo resumidas a partir das definições encontradas em (CARDOSO; VALETTE, 1997):

- **Limitabilidade:** Uma RP é considerada limitada se todos os lugares dessa rede possuem uma limitação máxima no número de fichas.
- **Vivacidade:** Uma RP é considerada viva se todas as suas transições podem ser acessadas a partir de qualquer marcação acessível M', através de uma sequência s de disparo.
- **Reiniciabilidade:** Uma RP é reiniciável apenas se for possível a partir de qualquer marcação acessível M' encontrar uma sequência de disparo s que leve de volta a marcação inicial.

2.6 Redes de Petri Coloridas

A rede de Petri Colorida (RPC) é uma linguagem gráfica para construir modelos de sistemas concorrentes e analisar suas propriedades. As redes de Petri são divididas em duas classes: RP de baixo nível e RP de alto nível. As RPC pertencem a classe de alto nível, pois combinam as capacidades de uma RP ordinária com as capacidades de uma linguagem de programação (JENSEN; KRISTENSEN, 2009).

Nas RPC, as fichas possuem um valor de dados anexado que são chamados de cores. Essas cores podem representar diferentes processos ou recursos de uma rede, reduzindo assim o tamanho dos modelos (MACIEL; LINS; CUNHA, 1996). Para um determinado lugar, todas as fichas devem ter cores que pertençam a um tipo específico que é chamado de conjunto de cores do lugar. Esse uso de conjunto de cores é equivalente aos tipos de dados nas linguagens de programação (inteiro, caractere, booleano) e a especificação dos conjuntos de cores e das variáveis da rede é feita por declarações (JENSEN, 1997) .

A estrutura da RPC é composta por um grafo dirigido com dois tipos de vértices (lugares e transições). Os lugares são representados por círculos ou elipses e as transições por retângulos. Cada lugar, transição e arco da rede possui inscrições associadas. Por exemplo, os lugares tem três tipos de inscrições: nomes, conjunto de cores e expressão de inicialização que é a marcação inicial (MACIEL; LINS; CUNHA, 1996).

Uma definição formal para uma RPC foi descrita em (JENSEN; KRISTENSEN, 2009) onde:

Definição 2. ***(Rede de Petri Colorida)***
Uma Rede de Petri Colorida é uma nove-tupla $RPCM = (P, T, A, \Sigma, V, C, G, E, I)$ *onde:*

1. P *é um conjunto finito de lugares.*
2. T *é um conjunto finito de transições* T *tal que* $P \cap T = \varnothing$.
3. $A \subseteq P \times T \cup T \times P$ *é um conjunto de arcos direcionados.*
4. Σ *é um conjunto finito de cores não vazio.*
5. V *é um conjunto de variáveis tipadas de tal forma que* $Type[v] \in \Sigma$ *para todas variáveis* $v \in V$.
6. $C : P \rightarrow \Sigma$ *é uma função de conjunto de cores que atribui um conjunto de cores a cada lugar.*
7. $G : T \rightarrow EXPR_V$ *é uma função de guarda que atribui um guarda a cada transição* t *tal que* $Type[G(t)] = Bool$.
8. $E : A \rightarrow EXPR_V$ *é uma função de expressão de arco que associa uma expressão de arco a cada arco* a *tal que* $Type[E(a)] = C(p)_{MS}$*, onde* p *é o lugar conectado ao arco* a.

9. $I : P \rightarrow EXPR_{\emptyset}$ *é uma função de inicialização que atribui uma expressão de inicialização para cada lugar* p *tal que* $Type[I(p)] = C(p)_{MS}$.

As diferentes marcações que uma RPC pode alcançar são chamados de espaço de estado (*state space*). Jensen e Kristensen (2009) definem um espaço de estado como um grafo direcionado no qual temos um nó para cada marcação acessível e um arco para cada elemento de ligação que ocorre. Eles também afirmam que um espaço de estado representa todas as execuções possíveis do sistema em consideração e pode ser usado para verificar, que o sistema possui uma determinada propriedade formalmente especificada.

2.7 CPN Tools

CPN Tools[1] é uma ferramenta para edição, simulação, análise de estados e análise de desempenho de modelos de RPC. Essa ferramenta possui uma ampla documentação com tutoriais que ajudam o usuário a entender o funcionamento detalhadamente. Além de ser uma ferramenta poderosa, o CPN Tools tem uma licença gratuita, o que favorece pesquisadores a desenvolver seus projetos com menor custo.

Na Figura 8 podemos visualizar a interface do programa. À esquerda temos o menu com uma série de ferramentas incluindo a de criação, simulação, monitoramento e gerenciamento de arquivos para carregar ou salvar uma rede. Quando se inicia um novo projeto mais opções aparecem, como um menu para declaração de variáveis e conjunto de cores. Na aba *Page* é onde a rede pode ser gerada.

No exemplo da Figura 8 a rede inicia em $g6$ contendo duas fichas que representam as cores $corX$ e $corY$, as duas transições $t1$ e $t2$ se encontram ativas, o que é indicado pela cor verde no contorno delas. Podemos ver na transição $t1$ uma expressão $b = corX$ que funciona como guarda para que apenas essa cor específica possa ativá-la. Quando a ficha vai para $g7$ ela muda de cor pela expressão de arco que transforma $corX$ em $corB$. Já a palavra *EXAMPLE* é o nome do conjunto de cores definido no menu lateral.

Uma das deficiências do programa é a falta da contagem de caminhos. Por esse motivo, usaremos as definições de caminho para um grafo direcionado que são apresentadas no Capítulo 4.

[1] http://cpnTools.org

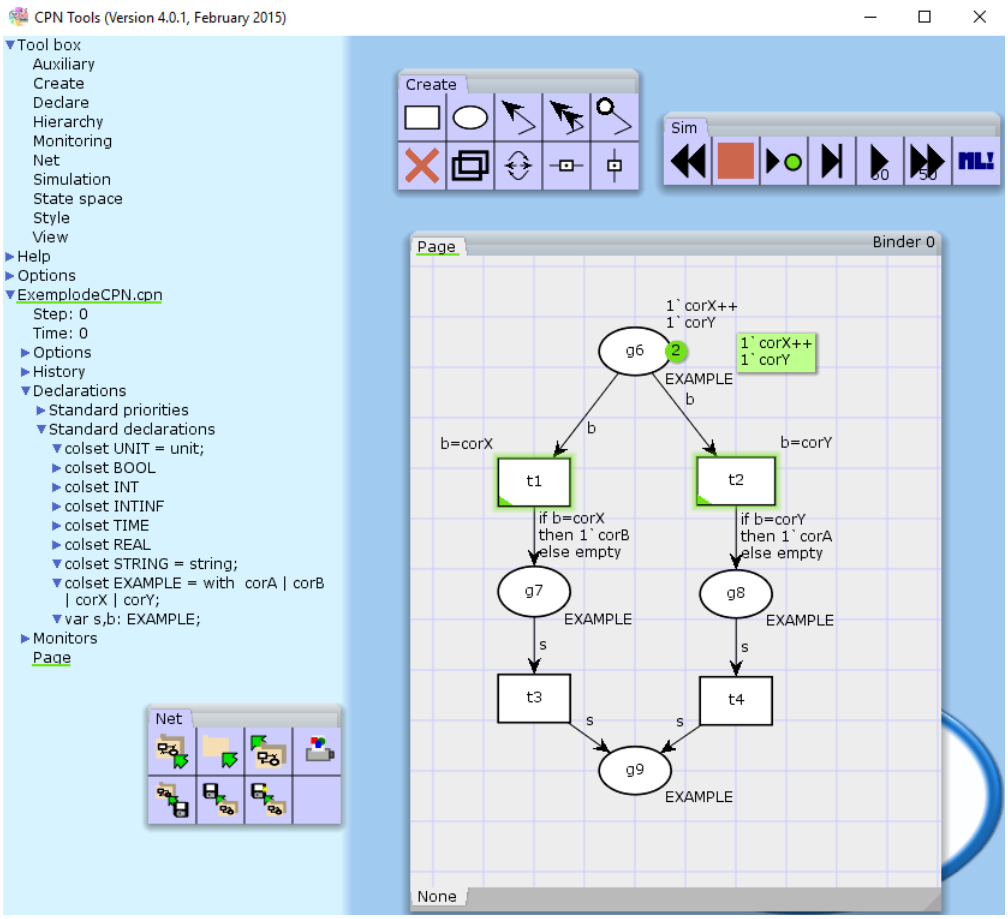


Figura 8: Interface do CPN Tools

3 TRABALHOS RELACIONADOS

A busca por trabalhos relacionados ao tema foi baseada em artigos que tivessem como foco o teste em sistemas multiagente. Ela foi orientada pelas seguintes palavras chaves: "*multi-agent system*", "*test case*", "*correction*", "*verification*", "*validation*", "*error*", "*faults*", "*failure*". O resultado destas buscas é relatado a seguir.

Kissoum e Sahnoun (2007) descrevem uma abordagem formal para especificar, desenvolver e testar SMA. Esta abordagem baseia-se na utilização da linguagem algébrica de especificação formal Maude (CLAVEL et al., 2002), que facilita as descrições e representações das relações entre os elementos de forma transparente no que diz respeito ao comportamento interno de cada classe de agentes. Também foi utilizado um conjunto de critérios de cobertura que podem orientar a geração de sequências de teste e calcular a eficiência da abordagem de teste. Vale ressaltar que os testes propostos no artigo estão restritos a interação do agente e seu comportamento interno.

Coelho et al. (2007) apresentam um *framework* para teste de agentes desenvolvido na plataforma JADE. A ferramenta realiza os testes utilizando agentes falsos (*mock agents*) para testar outro agente através do monitoramento do comportamento dele e além disso é gerado um cenário de testes para definir uma série de condições onde o agente em teste será exposto. A ferramenta é capaz de realizar testes de unidade, integração e sistema.

Nguyen et al. (2007) desenvolvem uma ferramenta para gerar casos de teste de forma automática chamada eCAT em que são apresentadas duas técnicas. A primeira é randômica, onde um agente autônomo de teste é capaz de gerar casos de testes de forma aleatória utilizando troca de mensagens com o agente em teste enquanto agentes de monitoramento analisam as reações. A segunda chamada de mutação evolutiva é uma combinação do teste de mutação, onde os operadores de mutação são aplicados ao programa original para introduzir artificialmente defeitos conhecidos, com o teste evolucionário onde suítes de teste são desenvolvidas aplicando operadores de mutação nos casos de teste.

Nguyen, Perini e Tonella (2008) definem uma abordagem para geração de casos de teste baseado em ontologia para servir como guia na geração de testes utilizando o *framework* desenvolvido anteriormente pelo mesmo autor. Um conjunto de regras de geração

foi definido e, usando-as, uma estrutura de teste automatizada pode testar extensivamente um determinado agente por meio de um grande e diverso número de casos de teste.

Gomez-Sanz et al. (2008) apresentam avanços em testes e depuração feitos pela metodologia INGENIAS (PAVÓN; GÓMEZ-SANZ; FUENTES, 2005). Baseado numa abordagem direcionada por modelos essa metodologia possibilita especificar os casos de teste durante a modelagem do sistema. O modelo é então transformado num código onde os testes podem ser executados e o desenvolvedor coletar informações relativas tanto às interações entre os agentes como ao estado mental deles em tempo de execução. Esse trabalho foca no teste do agente e das interações entre eles.

Salamon (2009) apresenta uma abordagem para teste de SMA com três camadas. Sendo a primeira camada responsável pela condução de teste de unidade em agentes. A segunda camada é referente aos testes das interações entre esses agentes onde erros como um *deadlock* podem ser detectados. Por fim a terceira camada constitui o teste do SMA como um todo onde gargalos no sistema ou outros problemas estruturais podem ser corrigidos.

Poutakidis et al. (2009) apresentam um *framework* para teste e depuração de SMA baseado no uso de artefatos de design. Mais precisamente foram demonstrados dois conceitos, um gerando artefatos de design para gerar casos de teste em testes de unidade e o outro usando esses artefatos para auxiliar na depuração do código.

Zhang, Thangarajah e Padgham (2009) descrevem uma abordagem para teste unitário baseado em planos de sistemas de agentes, com foco na geração automática de casos de teste. O *framework* se concentra em determinar a ordem na qual as unidades serão testadas, testar planos de agentes (unidades), incluir no código fonte do programa a ser testado mecanismos que possam gerar informações adicionais para o sistema de teste e por fim executar os casos de teste para coletar e analisar os resultados.

Miller, Padgham e Thangarajah (2010) definem critérios de cobertura para testes de interações em SMA. Esses critérios de cobertura de testes tem como finalidade medir a qualidade dos casos de teste que serão executados. Dois tipos de critérios são apresentados sendo um baseado apenas na especificação de protocolos de mensagem e o outro que leva em consideração também os planos dos agentes para a troca dessas mensagens. O artigo provê uma base para futuras especificações de geração de casos de teste projetados para fornecer uma boa cobertura.

Houhamdi e Athamena (2011) apresentam uma abordagem para testes estruturais em SMA especificando um processo de teste que complementa a metodologia orientada a metas chamada Tropos (GIUNCHIGLIA; MYLOPOULOS; PERINI, 2002). Nele é fornecida uma orientação sistemática para gerar conjuntos de testes a partir de artefatos de modelagem produzidos junto com o processo de desenvolvimento. Esses conjuntos de testes podem ser usados para refinar a análise de metas dos agentes e detectar problemas no início do processo de desenvolvimento.

Wang e Zhu (2012) propôs um *framework* de automação de testes chamado CATest utilizando a linguagem de especificação formal baseada em agentes SLABS (ZHU, 2001). Durante a execução do programa de teste o comportamento dos agentes é monitorado e gravado. Então esses comportamentos gravados são checados através de um verificador de correção e um verificador de adequação para garantir que estejam funcionando de acordo com a especificação. Uma limitação é a não existência de um gerador de casos de teste, eles precisam ser definidos manualmente pelo usuário, e a outra é que o sistema testa apenas o comportamento do agente individualmente.

Eassa et al. (2014) desenvolveram uma ferramenta de teste dinâmico que usa uma linguagem de asserção, declarativa, de lógica temporal para detectar erros em tempo de execução dos agentes. A linguagem foi uma proposta do próprio autor e através das declarações inseridas a ferramenta pode identificar erros dinâmicos durante a execução do programa de teste. Essa ferramenta gera agentes de teste para monitorar, controlar e gerar os testes que podem ser tanto a nível de agente quanto de integração.

Rehman e Nadeem (2015) apresentam uma abordagem para teste em SMA baseado em design de artefatos da ferramenta Prometheus (PADGHAM; THANGARAJAH; WINIKOFF, 2008). A ideia é gerar através de um diagrama de protocolos um gráfico de protocolos. Os dados deste gráfico juntamente com critérios de cobertura pré definidos servem de entrada para gerar caminhos de teste que cubram as interações entre os agentes. Esses caminhos de teste podem ser usados num trabalho futuro para uma geração automática de casos de teste.

Kerraoui et al. (2016) proporam uma abordagem de teste em SMA baseada em Modelos. Primeiramente é gerado e validado um modelo que represente o comportamento do sistema através de uma rede de Petri. Em seguida os casos de teste são gerados automaticamente tendo como entrada o modelo comportamental do sistema. Por fim uma versão instrumentada do modelo é gerada para a execução dos casos de teste e geração dos resultados. O modelo abrange os níveis de teste de agente, integração e sistema e uma das limitações é que o sistema consegue realizar apenas testes funcionais.

Nascimento et al. (2017) modelaram e desenvolveram uma arquitetura baseada em publicação de assinaturas para facilitar a implementação de sistemas que testem os SMA nos níveis de agente e grupo. Os autores utilizaram uma plataforma chamada RabbitMQ (RICHARDSON et al., 2012) para entregar os *logs* (publicações) dos agentes para serem utilizados por aplicações de teste (assinantes). Usando máquinas de estado os aplicativos de teste puderam validar esses casos de teste comparando os *logs* consumidos do editor MAS com os *logs* listados para validação. No final o desenvolvedor pode usar a interface para identificar a falha e reduzir o tempo de diagnóstico. Uma limitação citada é a falta de recursos para lidar com as características não-determinísticas dos sistemas multiagentes.

Winikoff (2017), com foco em programas de agentes BDI, analisa sua testabilidade em relação ao critério de adequação de testes de todas as arestas "*all edges*". Para isso

eles fazem uma comparação de quantidades de testes necessários para os critérios todas as arestas e todos os caminhos "*all paths*", e mostram que o número de testes necessários para cobrir todas as arestas é bem menor do que todos os caminhos. Esse artigo também faz uma comparação para mostrar o quanto programas BDI são mais difíceis de testar do que programas processuais de tamanho equivalente. Portanto ele realiza uma avaliação de certos critérios para que possam ser analisados e utilizados em futuras aplicações. Segundo o autor, as comparações que foram feitas podem ser representadas a nível de agente, partes de um agente (unidade) ou o sistema todo.

Barnier et al. (2017) apresentam uma nova estratégia para testes em sistemas multiagentes embarcados. Sua metodologia foca em três itens principais que são: teste de agente, teste de recursos coletivos e teste de aceitação. Em cada item os autores definem quais as metas que devem ser alcançadas para que o teste seja bem sucedido. Por exemplo, o teste de agente tem como metas os testes de software, hardware e integração do agente assim como a análise do tempo de resposta do mesmo.

Gonçalves, Rodrigues e Machado (2019) apresentam um método para avaliação da testabilidade de um SMA especificado sob o modelo organizacional $\mathcal{M}$oise com a proposta de um modelo estendido de Rede de Petri Colorida que dimensiona o número de testes necessários a validação de uma especificação $\mathcal{M}$oise^{+}. Primeiramente foi realizado um mapeamento entre o método para medir o grau de testabilidade de um programa de BDI proposto em (WINIKOFF, 2017) e sua correspondência quando especificado por rede de Petri. A seguir um método que adapta esse mapeamento para medir a testabilidade de uma especificação $\mathcal{M}$oise baseada em redes de Petri coloridas é apresentado. Por fim foi desenvolvido um software que transforma a descrição da rede de Petri colorida em um grafo direcionado, contando os caminhos dentro desse grafo e indicando o número de casos de teste necessários para validá-lo.

A Tabela 2 apresenta uma classificação das publicações de acordo com os níveis de teste para um SMA. Nela, podemos ter uma visão geral dos trabalhos na área de teste de SMA que foram publicados na última década. Como podemos visualizar alguns trabalhos abrangem em sua abordagem diferentes níveis de teste.

Nesse capítulo foi possível ter uma síntese de alguns trabalhos publicados relacionados com o tema de teste em SMA. O foco da maioria desses trabalhos foi apresentar uma metodologia para geração de casos de testes, porém nenhum deles leva em consideração um modelo organizacional do sistema e as restrições ao comportamento do agente que esse modelo pode proporcionar. Nesse sentido uma proposta que leva em conta esse quesito será apresentada no capítulo seguinte.

Unidade	(COELHO et al., 2007), (POUTAKIDIS et al., 2009), (ZHANG; THANGARAJAH; PADGHAM, 2009), (SALAMON, 2009), (WINIKOFF, 2017).
Agente	(KISSOUM; SAHNOUN, 2007), (NGUYEN et al., 2007), (GOMEZ-SANZ et al., 2008), (NGUYEN; PERINI; TONELLA, 2008), (WANG; ZHU, 2012), (EASSA et al., 2014), (KERRAOUI et al., 2016), (NASCIMENTO et al., 2017), (WINIKOFF, 2017), (BARNIER et al., 2017).
Integração	(COELHO et al., 2007), (SALAMON, 2009), (POUTAKIDIS et al., 2009),(GOMEZ-SANZ et al., 2008), (MILLER; PADGHAM; THANGARAJAH, 2010), (EASSA et al., 2014), (REHMAN; NADEEM, 2015), (KERRAOUI et al., 2016), (NASCIMENTO et al., 2017), (BARNIER et al., 2017), (GONÇALVES; RODRIGUES; MACHADO, 2019).
Sistema	(COELHO et al., 2007),(SALAMON, 2009), (HOUHAMDI; ATHAMENA, 2011),(KERRAOUI et al., 2016), (WINIKOFF, 2017).
Aceitação	(BARNIER et al., 2017)

Tabela 2: Classificação dos Artigos

4 TESTES EM ESPECIFICAÇÕES SOB O MODELO $\mathcal{M}$OISE$^+$

Como visto no capítulo anterior, diferentes abordagens com foco em testes de SMA foram listadas. O trabalho de Winikoff (2017) em especial foi utilizado como referência inicial para a elaboração da dissertação de Rodrigues (2018), à qual podemos considerar como primeira etapa da sequência que está sendo seguida nesse trabalho.

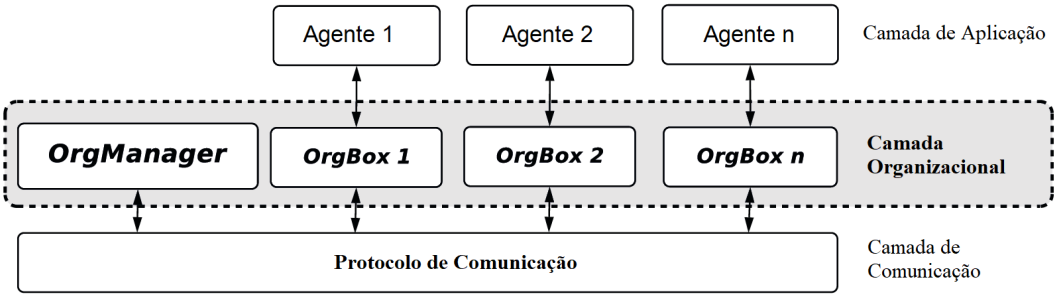


Figura 9: Componentes de um SMA utilizando um modelo organizacional, adaptado de (HÜBNER; SICHMAN; BOISSIER, 2005)

Na tentativa de mitigar a complexidade dos procedimentos de teste em Sistemas Multiagentes, propomos uma abordagem baseada numa perspectiva em que um SMA é visto a partir de três dimensões (GONÇALVES; QUADROS; SALDANHA, 2016): (i) a *organização*, que corresponde a descrição do nível social, onde se estrutura formal ou informalmente os elementos que viabilizam a interação e a coordenação entre os agentes; (ii) a *comunicação* que trata dos aspectos relacionados a protocolos e trocas de informação entre os agentes e; (iii) os *agentes* que correspondem a especificação do nível individual onde se instanciam as demandas da organização a partir das capacidades e habilidades individuais de cada agente. Essas dimensões podem ser vistas nas três camadas da Figura 9 que apresenta os diferentes componentes de um SMA centrado numa organização. Sob esta perspectiva, entende-se que Winikoff (2017) resolve os aspectos de testabilidade no nível dos agentes, ou trata da Camada de Aplicação segundo Hübner, Sichman e Boissier (2005), de modo que os níveis de comunicação e organização ainda demandam seu pro-

cedimentos e sistematização. Neste projeto de dissertação apresentamos uma abordagem para tratar sobre testes no nível de organização de um sistema multiagente.

Enquanto em (WINIKOFF, 2017) um grafo de controle de fluxo foi utilizado para avaliar a testabilidade de um sistema BDI, Rodrigues (2018) adapta esse grafo transformando-o numa RPC para servir como ferramenta de descrição e análise de uma organização $\mathcal{M}$oise$^+$. No final dessa análise um método para contagem de caminhos de teste em um SMA utilizando $\mathcal{M}$oise$^+$ é apresentada.

Além dos trabalhos acima, vale citar (ZHU; HE, 2002), que embora não tenha foco em sistemas multiagente, descreve uma metodologia de testes para redes de Petri de alto nível com foco em sistemas concorrentes e distribuídos. O método possui um sistema de observação e gravação dos fenômenos observáveis durante o teste de sistemas concorrentes utilizando quatro tipos diferentes de estratégias de teste: orientadas por transição, estado, fluxo e especificação. Para cada estratégia, um conjunto de esquemas de observação e gravação dos resultados dos testes e um conjunto de critérios de adequação de teste foram definidos.

Antes de descrever o método, é necessário explicar as razões para a escolha das redes de Petri como o principal recurso para este problema (GIRAULT; VALK, 2013): (i) elas fornecem um formalismo de modelagem fundamentado graficamente e matematicamente, que é um forte requisito para processos de desenvolvimento de sistemas que necessitem de ferramentas gráficas e algorítmicas; (ii) são capazes de integrar os diferentes níveis e tipos da especificação $\mathcal{M}$oise$^+$ como resultado de seus mecanismos de abstração e refinamento hierárquico; (iii) existe uma enorme variedade de algoritmos para a concepção e análise de redes de Petri e ferramentas poderosas foram desenvolvidas para ajudar neste processo; (iv) existem diferentes variantes de modelos de rede de Petri, todos relacionados pelo formalismo básico sobre o qual eles se baseiam. Além do modelo básico, a rede ordinária de Petri, existem extensões como redes temporizadas, estocástica, de alto nível, entre outras, atendendo às necessidades específicas de quase todas as áreas de aplicação.

A Figura 10 mostra uma visão geral das etapas contidas no modelo de metodologia de testes proposto nesse trabalho. A primeira etapa consiste em analisar uma especificação $\mathcal{M}$oise$^+$ pertencente ao SMA que será testado para na segunda etapa fazer um mapeamento dessa especificação em uma RPC. O mapeamento da rede consiste nos seguintes passos: (i) Gerar as Declarações, (ii) Modelar a Estrutura, (iii) Incluir as Inscrições, (iv) Incluir Transições de Falha. Com a RPC gerada a terceira etapa consiste em contar os casos de teste da rede, encerrando assim a parte de avaliação da testabilidade do $\mathcal{M}$oise$^+$. A quarta e última etapa será a geração de casos de teste relativo aos caminhos encontrados, esses casos de teste podem ser utilizados para a realização de testes no sistema. A seguir é feita uma descrição do método utilizado na proposta desse trabalho.

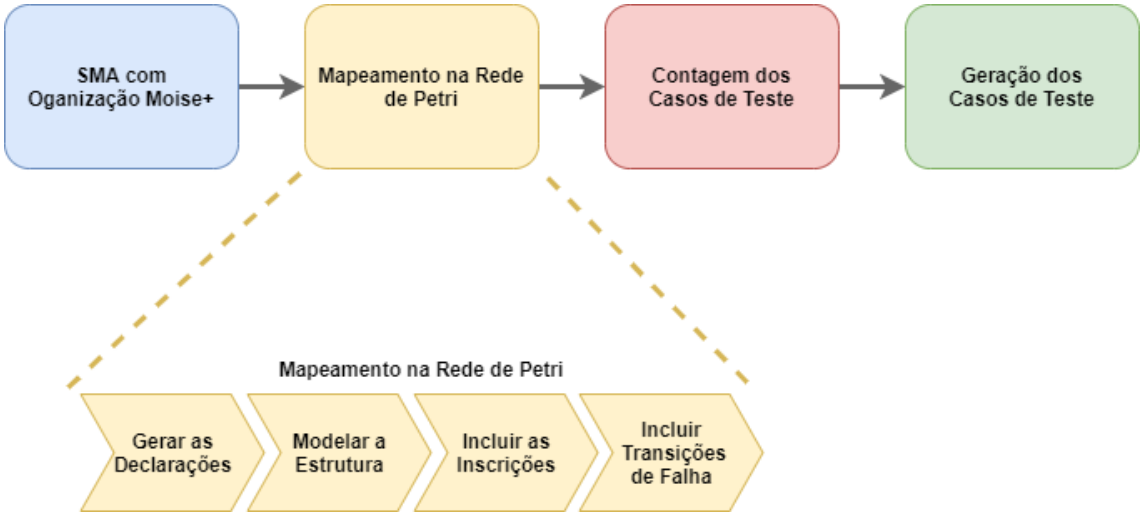


Figura 10: Sequencia de Etapas do Modelo Proposto

4.1 Definições de Rede de Petri para o $\mathcal{M}$oise$^+$

Como vimos, o método proposto por Winikoff (2017) utilizou um grafo de controle de fluxo para análise da testabilidade de um sistema BDI. Para nosso modelo, um grafo de controle de fluxo simples se torna inviável pois uma especificação $\mathcal{M}$oise$^+$ contém elementos como papéis, metas e planos que precisam ser especificados, na medida em que são instanciados pela dinâmica da organização. Como nas RPC é possível declarar e especificar cada elemento de sua arquitetura com uma etiqueta diferente, optamos por utilizar essa ferramenta para nosso mapeamento.

Em nosso método, as fichas coloridas são etiquetas empregadas na representação dos papéis da organização. Os lugares se associam ao conjunto de cores das fichas, ou seja, sendo os lugares metas. Então cada lugar terá a cor associada ao conjunto de papéis relacionados e de acordo com a especificação deôntica é possível correlacionar com a missão que representa aquele lugar. Os arcos que ligam os lugares às transições recebem agora uma variável ou uma expressão de arco com valor pertencente ao papel, descrevendo qual ficha está saindo do lugar X e qual ficha estará entrando no lugar Y.

A formalização deste mapeamento será proposta como uma rede de Petri Colorida para $\mathcal{M}$oise$^+$, baseada na definição de RPC em (JENSEN; KRISTENSEN, 2009), que será chamada de RPCpM.

Definição 3. ***(Rede de Petri Colorida para $\mathcal{M}$oise$^+$)***
Uma Rede de Petri Colorida para $\mathcal{M}$oise$^+$ (RPCpM) é uma nove-tupla $RPCpM = (P, T, A, \Sigma, V, C, G, E, I)$ *onde:*

1. P *é um conjunto finito de lugares com dimensão* n *que representa as metas numa organização* $\mathcal{M}$oise$^+$.

2. T *é um conjunto finito de transições com dimensão* m *que define o tipo de operador usado (sequência, escolha ou paralelismo) para executar os planos numa*

organização $\mathcal{M}$oise$^+$.

3. *A é um conjunto de arcos direcionados tal que $A \subseteq P \times T \cup T \times P$.*

4. *Σ é um conjunto de cores não vazio finito que representa papéis em uma organização $\mathcal{M}$oise$^+$.*

5. *V é um conjunto de variáveis tipadas de tal forma que $Type[v] \in \Sigma$ para todas variáveis $v \in V$.*

6. *$C : P \rightarrow \Sigma$ é uma função de conjunto de cores que atribui um conjunto de cores a cada lugar, define quais papéis são responsáveis por uma determinada meta em uma organização $\mathcal{M}$oise$^+$.*

7. *$G : T \rightarrow EXPR_V$ é uma função de guarda que atribui uma guarda a cada transição t tal que $Type[G(t)] = Bool$, é responsável por direcionar os papéis para suas respectivas metas numa organização $\mathcal{M}$oise$^+$.*

8. *$E : A \rightarrow EXPR_V$ é uma função de expressão de arco que associa uma transformação de cor a cada arco a tal que $Type[E(a)] = C(p)_{MS}$, onde p é o lugar conectado ao arco a, essa função é responsável pela verificação dos papéis necessários para a realização das metas numa organização $\mathcal{M}$oise$^+$.*

9. *$I : P \rightarrow EXPR_\emptyset$ é uma função de inicialização que atribui uma expressão de inicialização para cada lugar p tal que $Type[I(p)] = C(p)_{MS}$, define a ordem de execução para uma sequência de metas numa organização $\mathcal{M}$oise$^+$.*

4.2 Mapeamento na Rede

A primeira etapa de nosso método é o mapeamento na RPC. Utilizando a Definição 3 é possível gerar um mapa em RPC para um modelo organizacional $\mathcal{M}$oise$^+$. As etapas do mapeamento estão descritas na Figura 10, sendo constituída pelos seguintes passos: (i) Gerar as Declarações, (ii) Modelar a Estrutura, (iii) Incluir as Inscrições, (iv) Incluir Transições de Falha. Cada um desses passos é descrito a seguir.

Gerar as Declarações: As declarações são baseadas na Especificação Estrutural do modelo $\mathcal{M}$oise$^+$. Para cada papel da EE será gerada uma cor específica correspondente. Por exemplo, se numa EE temos um papel chamado *writer*, podemos criar uma cor chamada também de *writer* e depois atribuir essa cor a um conjunto de cores. Além das cores também é preciso declarar as variáveis que são usadas para retratar essas mesmas cores nos arcos. Abaixo temos um exemplo de declaração no CPN Tools. Podemos ver as cores *writer* e *editor* que pertencem a um conjunto de cores *Book* seguido por duas variáveis *w* e *e* baseadas nas cores do conjunto *Book*.

```
colset Book = with writer | editor ;
var w, e: Book;
```

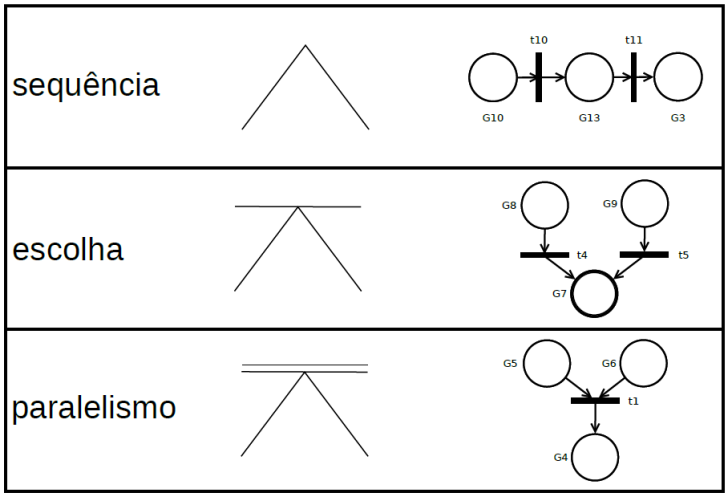


Figura 11: Relação dos operadores para uma rede de Petri (RODRIGUES, 2018).

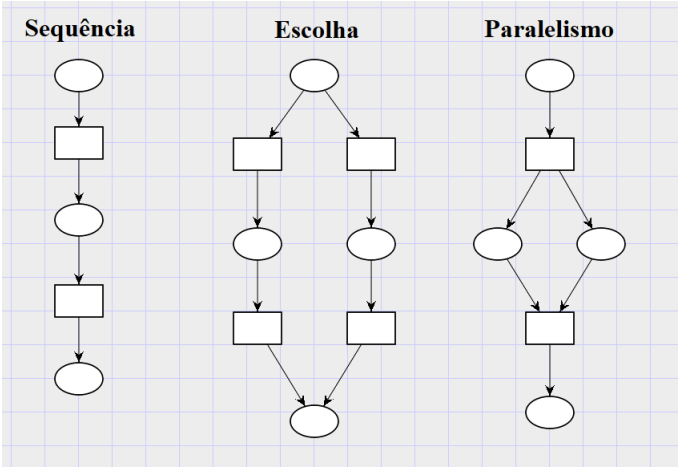


Figura 12: Exemplos de estruturas da RPCpM.

Modelar a Estrutura:

A modelagem da estrutura da RPC deve ser baseada na especificação funcional do modelo $\mathcal{M}$oise^{+}. Usando a relação de operadores descrito na Figura 11 é possível transformar cada operador de plano (sequência, escolha e paralelismo) de uma EF numa estrutura de RP correspondente. Na Figura 12 é possível visualizar cada um dos operadores transformados em estruturas da RPCpM no CPN Tools.

Incluir as Inscrições:

As inscrições RPC devem obedecer à atribuição de papéis a cada missão, que define quais cores devem ser atribuídas a cada lugar na rede, de acordo com a ED. A mesma lógica deve ser usada para definir transições e inscrições de arco. Neste ponto, já é possível realizar algumas simulações e já está definida a relação completa entre a EE e a EF.

Na Figura 13 podemos ver uma estrutura com inscrições incluídas. Essas inscrições são: as cores *writer* e *editor* que pertencem ao conjunto de cores *PAPEL*, três variáveis b,

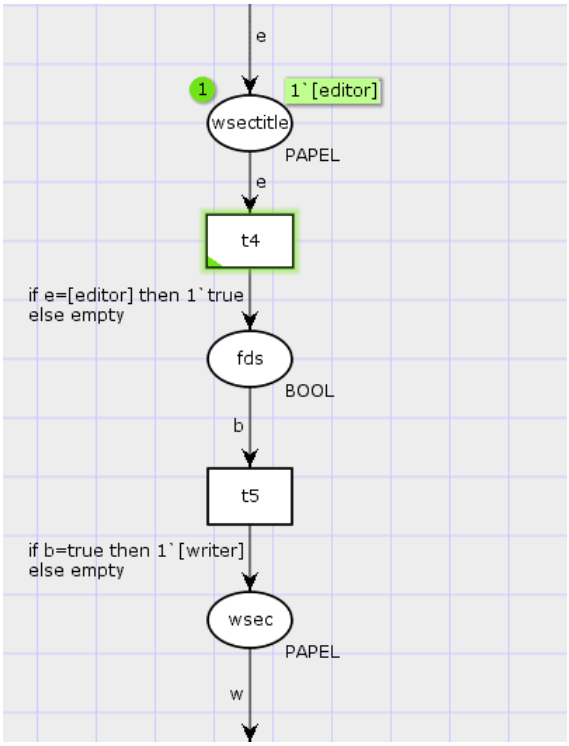


Figura 13: Estrutura da RPCpM com inscrições.

w e e, os lugares $wsectitle$, fds e $wsec$, as transmissões $t4$ e $t5$ e as funções de expressão de arco nas saídas de $t4$ e $t5$. Metas que não possuam uma missão vinculada e, portanto não necessitam de um papel que as satisfaça, dependendo apenas da conclusão de outras metas para satisfaze-las, recebem o tipo de cor $BOOL$ que representa o tipo booleano que aceita dois tipos de ficha: $true$ e $false$.

Inclusão de Falhas: A última parte do mapeamento da rede é a inclusão de transições que serão ativadas caso algum tipo de falha ocorra durante a execução de uma meta. Uma meta pode falhar pois as ações necessárias para a conclusão do plano, que o agente precisa executar para satisfazer uma determinada meta, podem não ocorrer como planejado. Tanto (WINIKOFF, 2017) como (HÜBNER, 2003) citam a possibilidade de falha na execução de um plano devido a uma interferência do ambiente.

Para a inclusão do tratamento de falha na RPC é necessário incluir um lugar denominado *fail*, onde todos os lugares com metas que possuem uma missão, não sendo portanto booleanos, da rede recebem uma transição para *fail*. Na Figura 14 podemos visualizar que a inclusão do tratamento de falha na rede, possibilitando que tanto as metas $wsectitle$ como $wsec$ possuam transições, $t12$ e $t14$, que representam a ocorrência de falha na execução da meta.

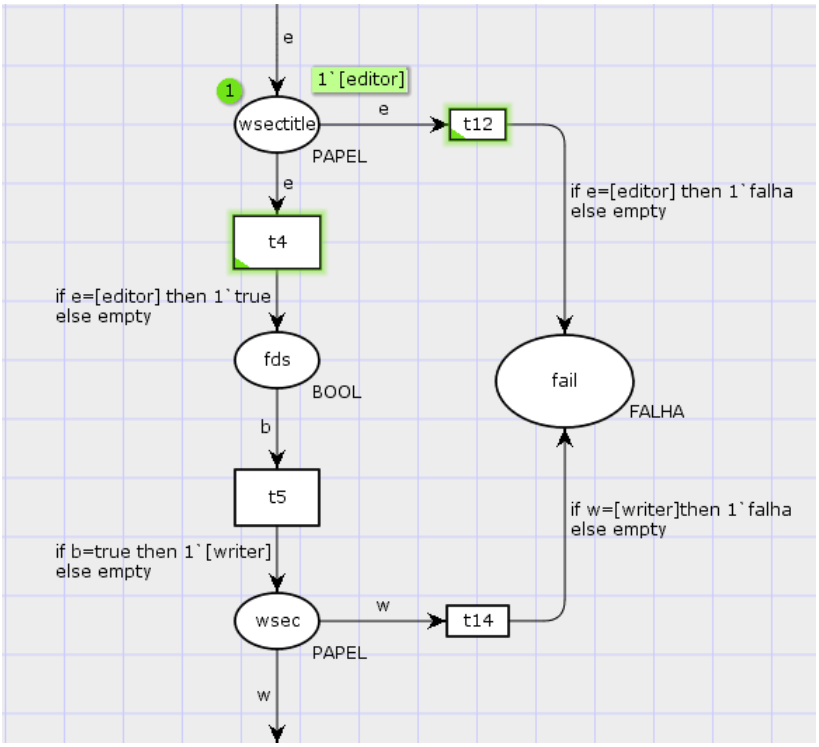


Figura 14: Estrutura da RPCpM com tratamento de falha.

4.3 Testabilidade - Contagem dos Casos de Teste

Terminado o mapeamento da rede, a próxima etapa do processo consiste na avaliação da testabilidade do modelo. Para compreender melhor o significado do termo testabilidade, vamos citar Winikoff (2017) que definiu testabilidade de um programa como sendo uma métrica que indica o esforço necessário para testar adequadamente esse programa. Por exemplo, para um programa bem simples como "**if** x **then** $t1$ **else** $t2$", são necessários dois testes para cobrir todos os caminhos de um grafo de controle de fluxo correspondente a esse programa. Antes de prosseguir vamos apresentar aqui as definições de grafo direcionado e caminho apresentadas em (CARDOSO; VALETTE, 1997), onde:

Definição 4. ***(Grafo Direcionado)*** *Um grafo direcionado $G = (V, E)$ consiste em um conjunto V de nós e um conjunto E de arcos tal que cada arco $e \in E$ é associado com um par ordenado de nós.*

Definição 5. ***(Caminho)*** *Considerando v_o a v_n nós em um grafo. Um caminho de v_o a v_n de comprimento n num grafo direcionado é uma sequência alternada de $n + 1$ nós e n arcos começando em v_o e terminando em v_n.*

Para avaliar a testabilidade de nosso modelo, será realizada a contagem dos casos de teste utilizando um critério de adequação de teste. Como visto na seção 2.3.1 existem diferentes critérios de adequação que podem ser utilizados. Esses critérios definem a forma como os casos de teste serão extraídos do sistema a ser testado. Em (WINIKOFF,

2017) os critérios utilizados para avaliação foram o de **todas as arestas** (*all edges*) e **todos os caminhos** (*all paths*). Rodrigues (2018) também utilizou o critério de adequação de **todos os caminhos**, porém sem levar em consideração os diferentes papéis que podem estar habilitados para executar uma determinada meta.

Vale lembrar que as RPC possuem uma complexidade maior que um simples grafo de controle de fluxo, pois trabalham com diferentes tipos de dados. Para exemplificar vamos analisar um modelo bem simples apresentado na Figura 15, onde na Figura 15a temos a ficha $roleA$ no lugar $goal1$, porém ao fazer a transição de $goal1$ para $goal2$ existem duas possibilidades de ficha que podem ser escolhidas, $roleB$ (Figura 15b) ou $roleC$ (Figura 15c) e por fim a ficha retorna a ser $roleA$ quando executa a transição $t2$ para o lugar $goal3$ (Figura 15d). Analisando esse exemplo na perspectiva de uma especificação $\mathcal{M}$oise$^+$, significa que o papel $roleA$ executa a meta $goal1$ e, ambos os papéis $roleB$ e $roleC$ são habilitados para realizar a meta $goal2$ porém somente um deles será escolhido, e no final o papel $roleA$ termina a execução com a meta $goal3$. Esse exemplo foi simplificado apenas para ilustrar uma situação específica e portanto não foram incluídas transições de falhas.

Se utilizarmos a Definição 5 para a contagem de caminhos no exemplo da Figura 15 teremos apenas um único caminho. Porém, como vimos, existem duas possibilidades diferentes de execução. Conclui-se que o critério de adequação **todos os caminhos**, embora seja um forte critério para testes em grafos de controle de fluxo, não consegue prever todas as possibilidades de execução para nosso modelo de RPC. Para tentar resolver essa lacuna, será utilizada uma abordagem diferente de critério de adequação baseado no critério chamado de **caminho de transição dos estados** (*state transition path*), que pode ser visto em (ZHU; HE, 2002) onde foi definido juntamente com o critério **todos os estados** (*all states*).

Definição 6. ***(Caminho de Transição dos Estados)*** *Sendo T um conjunto de execuções de teste, dizemos que T satisfaz o critério* ***caminho de transição de estados****, se e apenas se, para qualquer caminho de transição de estados alcançável p existe uma execução t em T tal que t cubra o caminho p.*

Estado	Lugar	Ficha
1	goal1	roleA
2	goal2	roleB
3	goal2	roleC
4	goal3	roleA

Tabela 3: Estados de Espaço para a RPC da Figura 15

Como vimos na seção 2.6, as diferentes marcações de uma RPC podem ser transformadas em um grafo direcionado chamado de espaço de estados. Nesse grafo os nós representam as marcações da rede e os arcos representam os elementos de ligação entre

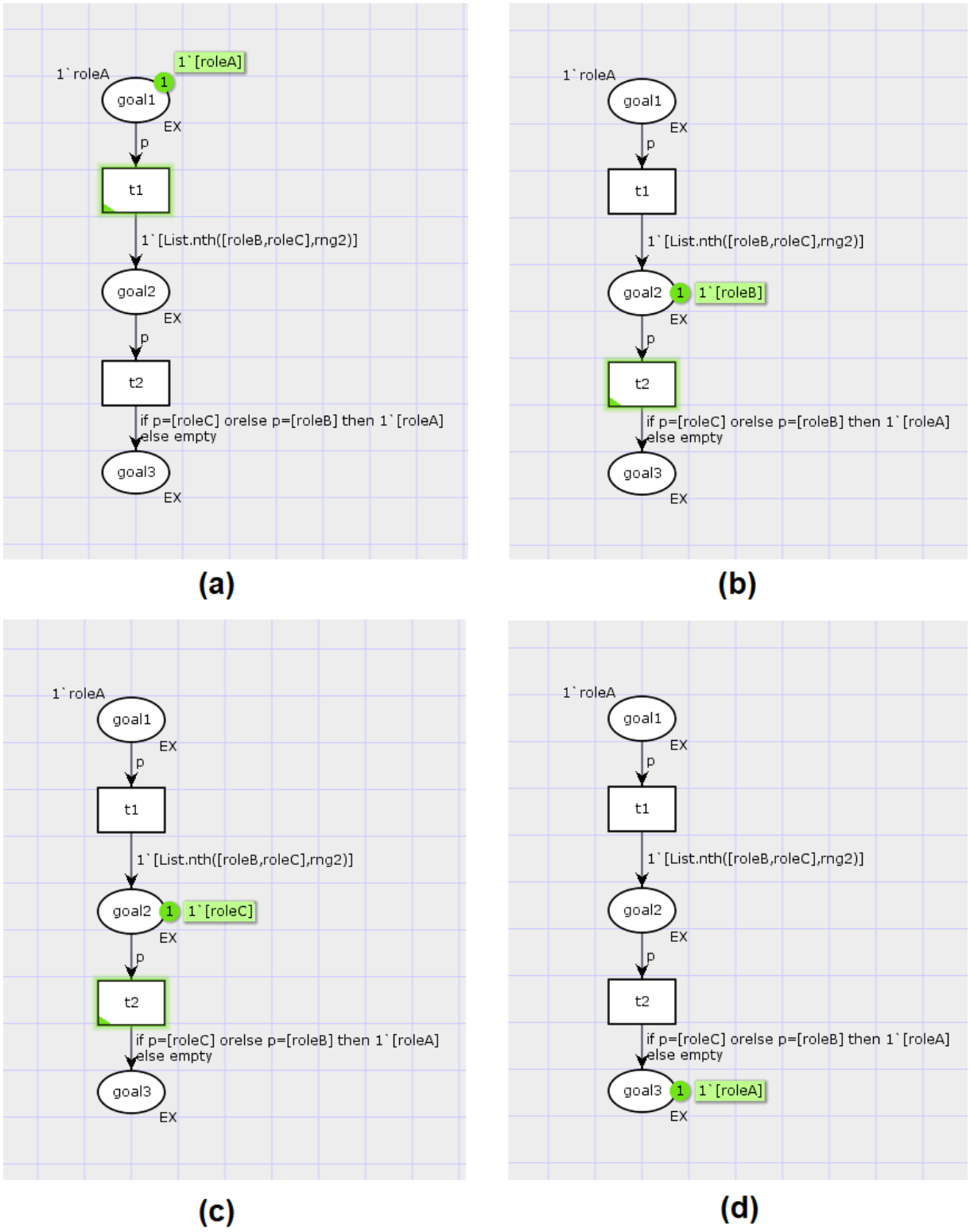


Figura 15: Exemplo de execução para uma RPCpM com diferentes possíveis papéis para uma mesma meta.

essas marcações. O critério **caminho de transição dos estados** leva em consideração todos os possíveis caminhos de execução para um grafo de espaço de estado numa RPC, sendo assim devemos identificar cada um desses caminhos.

Na Figura 17 é feito uma análise do espaço de estado para o nosso exemplo. Foram utilizadas funções do programa CPNTools, chamadas de $NoOfNodes()$ que retorna o número de nós, e $print(NodeDescriptor\ n)$ que mostra a marcação da rede para um determinado nó n. Mas antes de validar as funções é necessário clicar no botão *Calculate State Space*. Como vemos, existem quatro diferentes estados (nós) sendo eles apresentados na Tabela 3. Com essas informações e analisando o fluxo da rede que gera esses diferentes estados podemos gerar os caminhos que interligam cada um deles formando assim o grafo de controle de fluxo para o espaço de estado dessa rede (Figura 16).

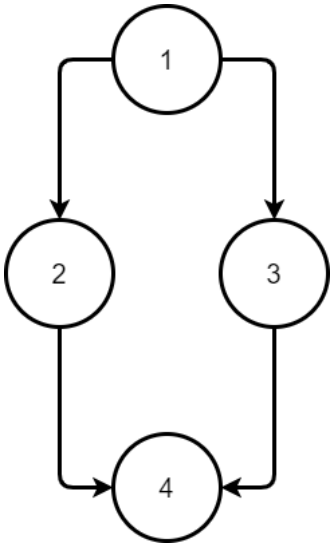


Figura 16: Grafo do Estado de Espaços para a RPC da Figura 15.

Analisando o grafo da Figura 16 temos dois diferentes caminhos de estados que ligam o estado 1 ao estado 4 que são:

1. $1 \rightarrow 2 \rightarrow 4$.
2. $1 \rightarrow 3 \rightarrow 4$.

Esses dois caminhos de estados representam as duas possíveis execuções para essa rede e, baseado no critério **caminho de transição dos estados**, serão os dois casos de teste utilizados para avaliação da testabilidade desse modelo.

4.4 Geração dos Casos de Teste

O método descrito acima serve para identificar e quantificar o número de casos de teste necessários para testar nosso modelo. Porém ainda é necessário especificar o cenário de teste que compõe cada caso. Assim, o objetivo dessa subseção será propor um método para especificar detalhadamente cada um desses casos de teste. Mas antes de descrever o método é preciso definir o que será testado e a maneira como serão realizados esses testes.

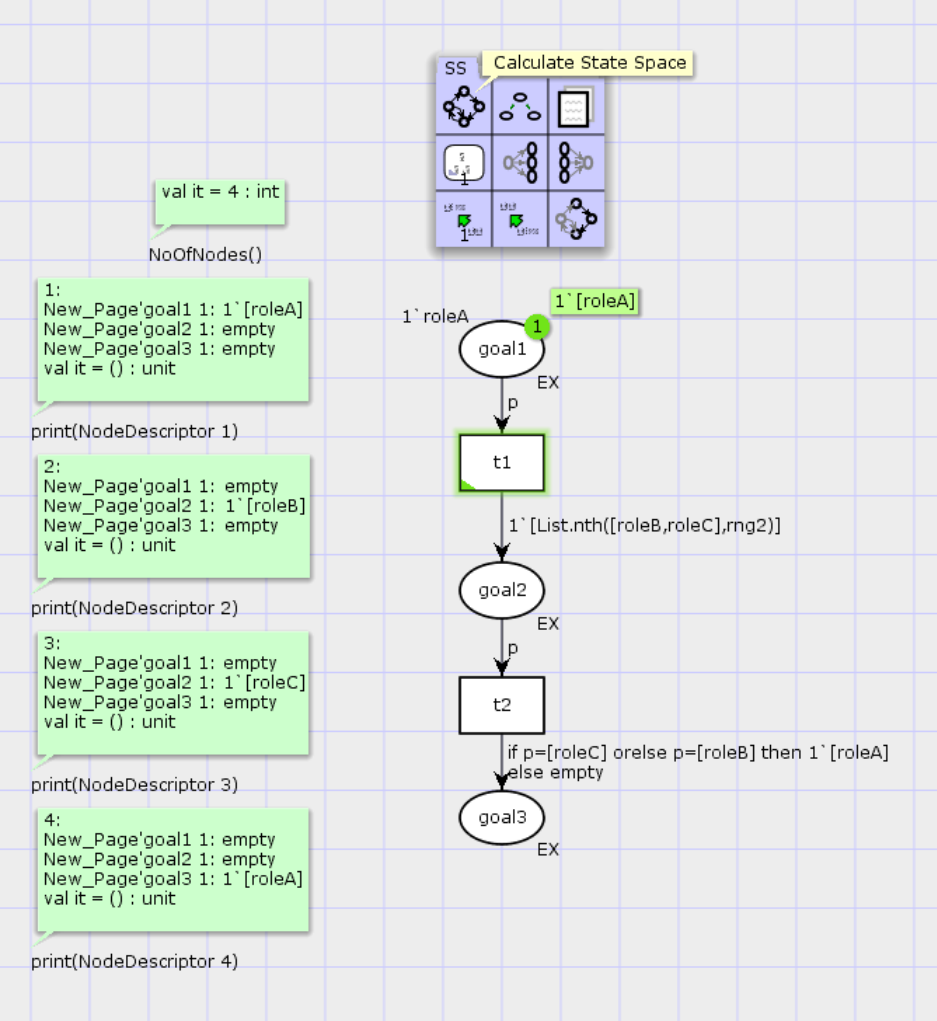


Figura 17: Identificação do espaço de estado para uma RPC

O que será testado: O teste do sistema será realizado para cada diferente caminho de teste que juntos formam um conjunto de teste. Cada caminho é uma sequência de execução de metas e são essas metas, ou seja as ações necessárias para cumpri-las que serão testadas.

Como será testado: No ambiente de teste serão realizadas simulações onde um agente com o papel x estando comprometido com a meta y deve executar ações que satisfaçam o plano necessário para cumprir a meta y seguindo a sequência do caminho de metas até o final ou até apresentar algum erro.

Os casos de teste contabilizados devem ser descritos individualmente para serem utilizados como guia na execução de testes do sistema. Cada caminho de teste devidamente identificado pelo método já apresentado, será um caso de teste. A descrição do caso de teste é feita através de uma tabela contendo o número do nó, a meta, papel e uma breve descrição dessa meta. Com essa tabela o caso de teste estará montado e pronto para ser utilizado como guia na execução do teste no sistema. Para nosso exemplo a Tabela 4 apresenta um caso de teste gerado para o caminho número 1 do grafo da Figura 16, enquanto

a Tabela 5 é referente ao caminho 2.

Nó	Meta	Papel	Descrição
1	$goal1$	$roleA$	O papel $roleA$ realiza a tarefa X
2	$goal2$	$roleB$	$roleB$ realiza a tarefa Y
4	$goal3$	$roleA$	$roleA$ finaliza com a tarefa Z.

Tabela 4: Exemplo de caso de teste caminho 1

Nó	Meta	Papel	Descrição
1	$goal1$	$roleA$	O papel $roleA$ realiza a tarefa X
2	$goal2$	$roleC$	$roleC$ realiza a tarefa Y
4	$goal3$	$roleA$	$roleA$ finaliza com a tarefa Z.

Tabela 5: Exemplo de caso de teste caminho 2

Os exemplos apresentados nesse capítulo serviram apenas para dar uma noção básica de como funciona o mapeamento da RPCpM sendo eles exemplos simples. No próximo capítulo um estudo de caso será apresentado mostrando cada passo do método na geração da rede para um modelo organizacional $\mathcal{M}$oise$^+$ específico e nele poderemos ver com mais detalhes cada uma dessas etapas aqui descritas.

5 CASOS DE USO

Para demonstrar como funciona o método proposto na seção anterior e sua capacidade de avaliar a testabilidade da especificação $\mathcal{M}$oise$^+$, serão utilizados dois exemplos de cenários de aplicação. O primeiro é o *Writing Paper* (HÜBNER; BOISSIER; BORDINI, 2011) já apresentado na seção 2.2, onde o objetivo final é escrever um artigo, amplamente usado na literatura e conhecido pela comunidade no assunto desta dissertação. O outro cenário é chamado de *Soccer Team* apresentado em (HÜBNER; SICHMAN; BOISSIER, 2002). Esse cenário representa um time de futebol que tem como meta principal marcar um gol. A escolha desse cenário se deve a maior complexidade da especificação, pois ele contém todos os tipos de operadores de planos do $\mathcal{M}$oise$^+$ o que possibilita apresentar o método de uma forma mais completa.

5.1 Write Paper

5.1.1 Descrição do Cenário

A Especificação Estrutural do cenário *Write Paper* pode ser vista na Figura 18, onde temos dois papéis $writer$ (escritor) e $editor$ (editor) que são sub-papéis de autor. Nesse cenário o objetivo é escrever um artigo.

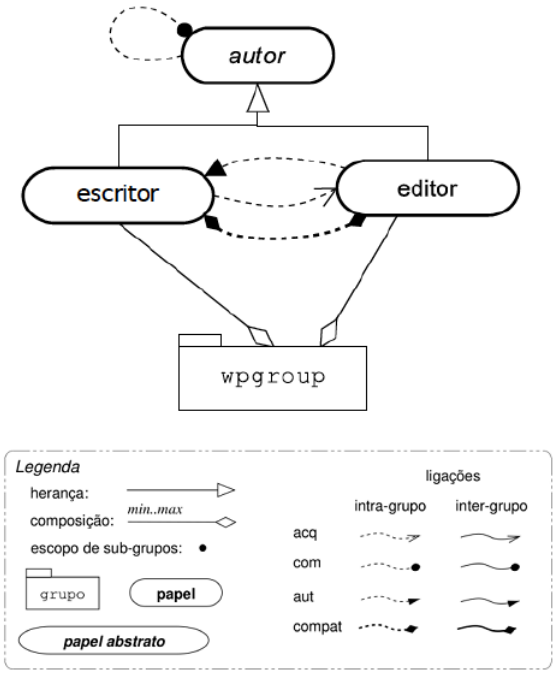


Figura 18: Especificação Estrutural do cenário *Write Paper* (HÜBNER; BOISSIER; BORDINI, 2011).

A EF se encontra na Figura 19, nela podemos notar que as metas $wtitle$, $wabs$, $wsectitle$ e $wcon$ pertencem a missão $mMan$, $wsec$ pertence a $mCol$ e $wref$ a $mBib$. Na Tabela 6 foi gerada uma descrição para cada uma das metas que compõem a EF.

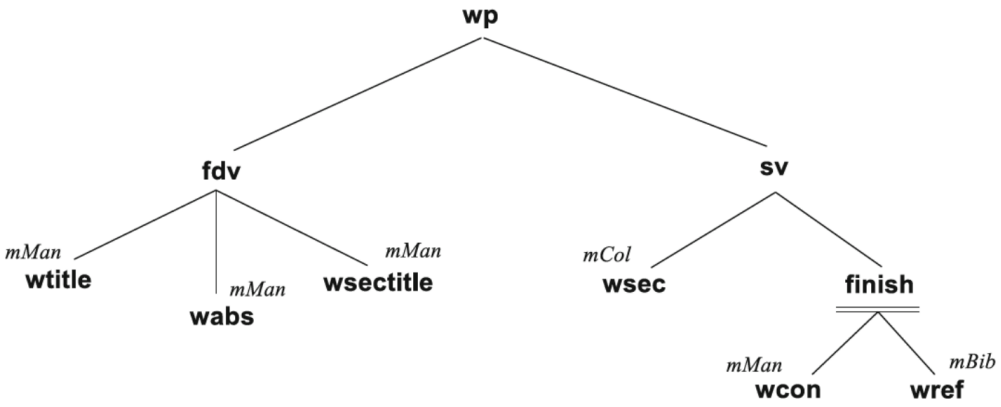


Figura 19: Especificação Funcional do cenário *Write Paper*(HÜBNER; BOISSIER; BORDINI, 2011).

Na Tabela 1 vemos a Especificação Deôntica onde o papel editor tem permissão para realizar a missão $mMan$ e o papel $writer$ tem obrigação para realizar tanto $mCol$ como $mBib$. Na Figura 5 é possível ver quais metas pertencem a $mMan$, $mCol$ e $mBib$.

meta	descrição
wp	Artigo foi escrito
fdv	Primeira versão do rascunho escrita
sv	Segunda versão escrita
wtitle	Um título foi escrito
wabs	Um resumo foi escrito
wsectitle	Os títulos das seções foram escritos
wsec	As seções foram escritas
wcon	A conclusão foi escrita
wref	As referências bibliográficas foram escritas
finish	Ambas conclusão e referências foram escritas

Tabela 6: Descrição das metas para o exemplo *Write Paper*

5.1.2 Implementação

Após a descrição do cenário vamos apresentar a implementação do método seguindo a sequência já apresentada:

5.1.2.1 Inclusão das Declarações

A primeira etapa é a declaração dos papéis e variáveis no CPNTool. Os papéis $writer$ e $editor$ pertencem ao conjunto de cores $PAPEL$. As variáveis w e e também serão declaradas, são elas as responsáveis por conduzir os papéis pelos arcos.

```
colset PAPEL = with writer | editor ;
var w, e: PAPEL;
```

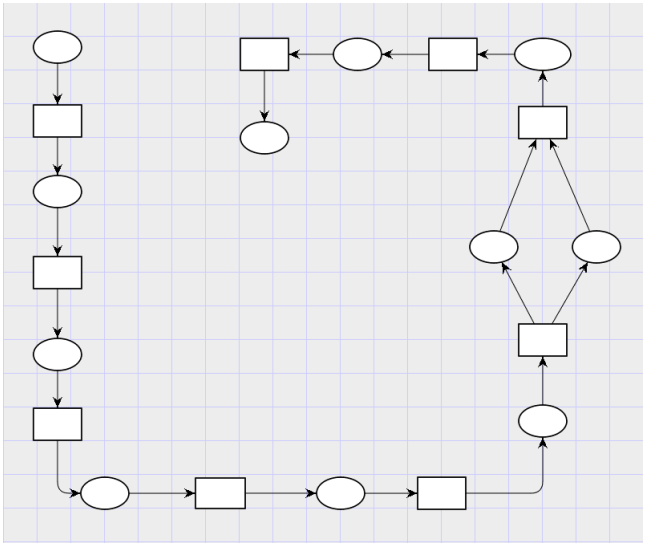


Figura 20: Estrutura da RPC para o Exemplo *Write Paper*

5.1.2.2 *Geração da Estrutura*

Com as declarações feitas a próxima etapa é gerar uma estrutura da rede utilizando os operadores de planos juntamente com a árvore de decomposição de metas da EF da Figura 5. Na Figura 20 podemos visualizar a estrutura da rede.

5.1.2.3 *Inclusão das Inscrições*

Essa etapa constitui na inclusão das inscrições na rede conforme podemos visualizar na Figura 21 onde os lugares representam as metas e as fichas identificam os papéis. A inclusão das inscrições corresponde a: colocar as expressões de arco para mudança de papéis; variáveis dos arcos; identificação dos lugares com nome da meta e o tipo de cor, por exemplo a meta $wtitle$ com o tipo $PAPEL$; além de uma identificação diferente para cada transição. O lugar chamado $start$ que é o primeiro da rede não constituí uma meta da especificação, ele é apenas uma representação de um estado inicial onde o sistema se encontra em espera para iniciar a execução.

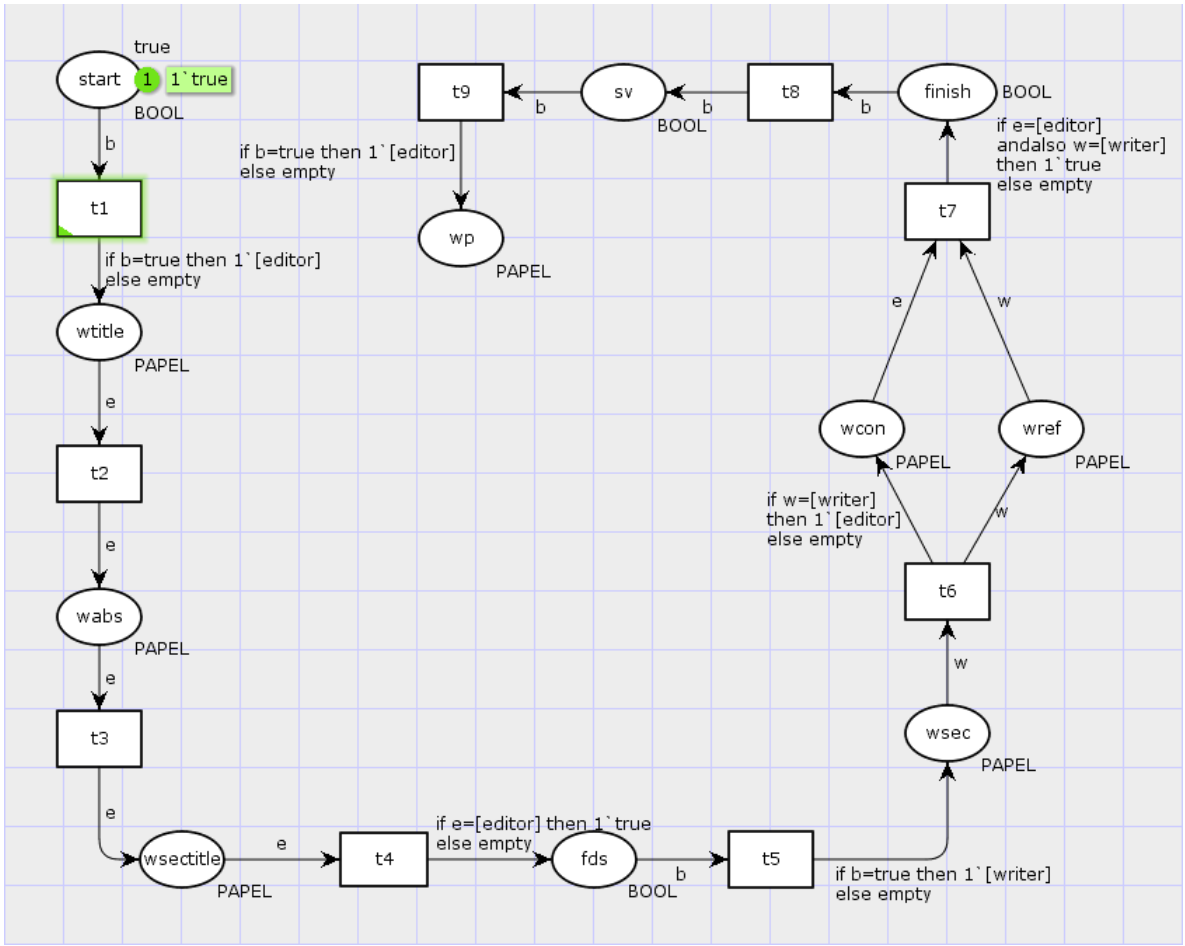


Figura 21: *Write Paper* no formato RPCpM com as Inscrições

5.1.2.4 *Inclusão das Falhas*

Para a inclusão dos caminhos de falha serão considerados todos os lugares da rede que não sejam do tipo $BOOL$, ou seja, sobrando apenas as metas que possuem uma missão vinculada e portanto um papel para realizar ações que satisfaçam essa meta. Nesses lugares, um caminho de falha será criado para um mesmo estado $fail$, indicando que uma falha ocorreu durante a execução do programa. Nesse exemplo conforme podemos visualizar na Figura 22 existem cinco diferentes caminhos de falha.

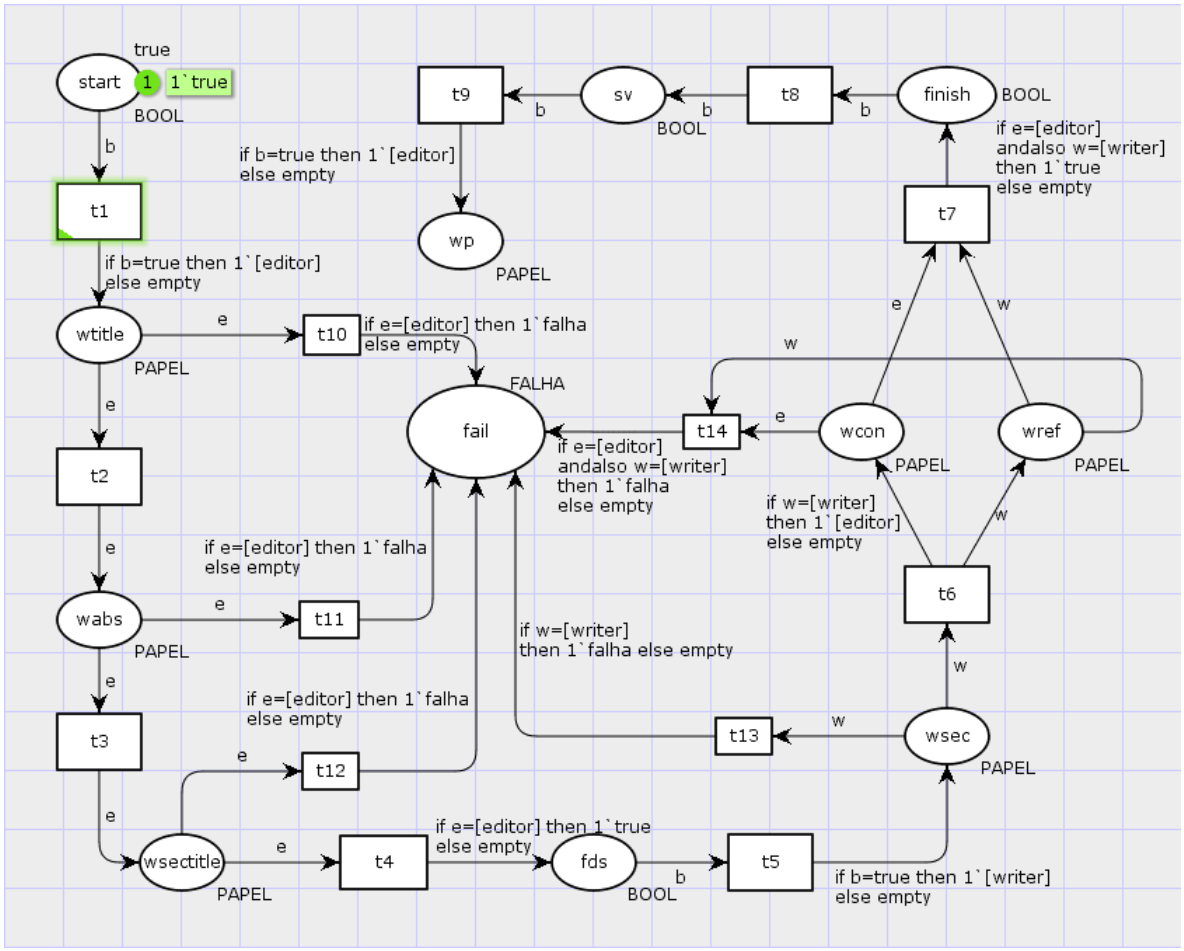


Figura 22: Write Paper no formato RPCpM

5.1.2.5 *Contagem dos Casos de Teste*

Nessa etapa serão contabilizados os caminhos de teste do sistema utilizando o critério de adequação **caminho de transição dos estados**. O primeiro passo será a identificação de todos os nós de estado que compõem os espaços de estado da rede. A ferramenta CPNTools possui algumas funções que podem facilitar esse processo. Na Figura 23 é possível ver essas funções chamadas de $NoOfNodes()$ que retorna o número de nós do espaço de estado sendo que essa rede possui onze nós e $print(NodeDescriptor\ n)$ que mostra a marcação da rede para um determinado nó n. Para que essas funções retornem

algum valor é necessário primeiro clicar no botão *Calculate State Space*.

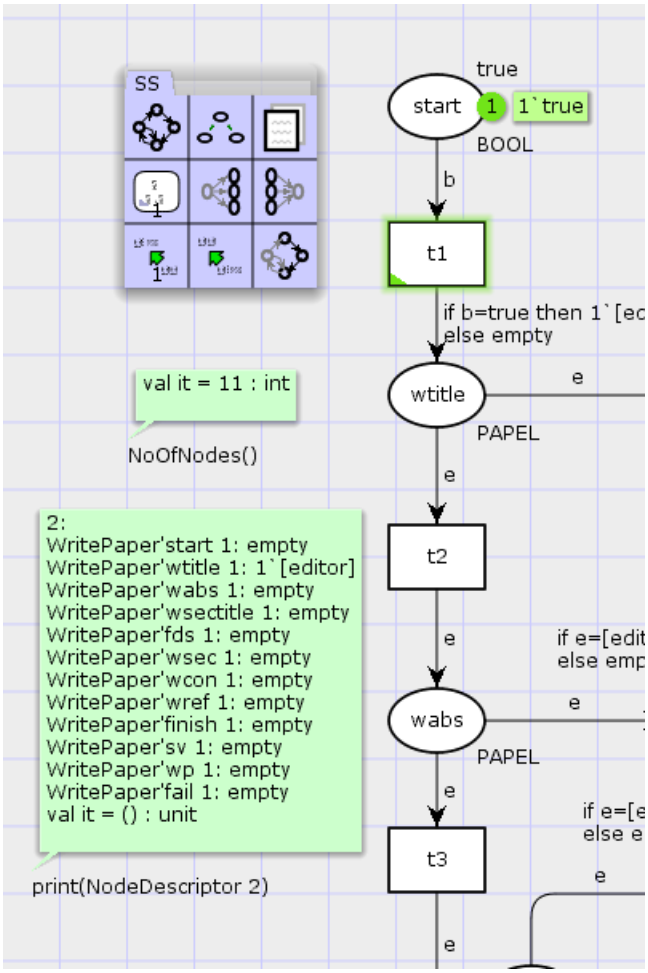


Figura 23: Contagem dos espaços de estados para o exemplo Write Paper.

Ainda na Figura 23 vemos descrição para o nó de estado número 2 da rede *Write Paper* onde a ficha $editor$ se encontra no lugar $wtitle$ e todos os outros lugares da rede se encontram vazios (*empty*). Para visualizar todos os possíveis estados dessa rede será necessário utilizar a função $print(NodeDescriptor\ n)$ para todos os 11 nós do espaço de estado de nosso exemplo.

Na Tabela 7 temos as marcações para cada estado mostrando apenas os lugares com fichas. No estado 8 temos dois lugares e duas fichas o que representa o estado onde o paralelismo ocorre.

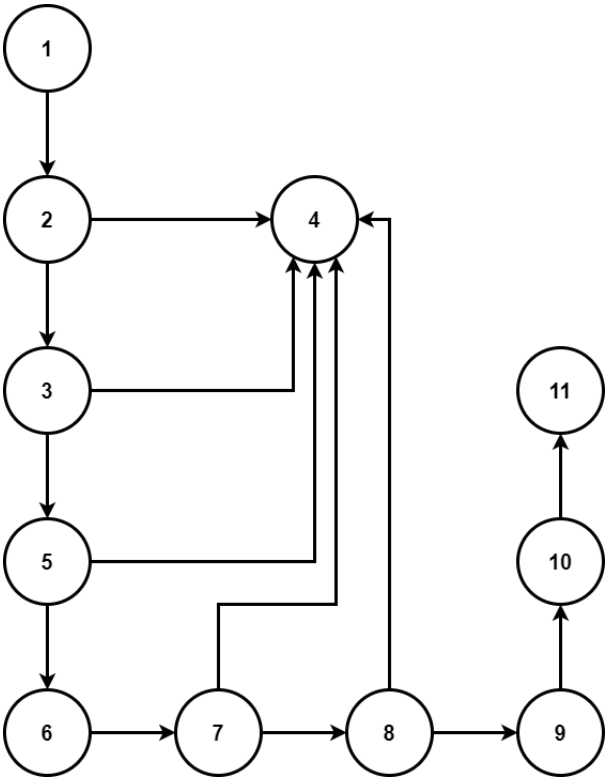


Figura 24: Diagrama do Espaço de Estados do *Write Paper*.

Estado	Lugar	Ficha
1	start	true
2	wtitle	editor
3	wabs	editor
4	falha	falha
5	wsectitle	editor
6	fds	true
7	wsec	writer
8	wcon,wref	editor,writer
9	finish	true
10	sv	true
11	wp	true

Tabela 7: Espaços de Estado do *Write Paper*

Gerada a tabela do espaço de estados da rede vamos agora gerar um grafo de controle de fluxo que mostra a ligação entre esses estados utilizando como referência a nossa RPCpM da Figura 22. Na Figura 24 se encontra o grafo com os onze estados e os caminhos que os ligam, nele fica fácil identificar os cinco caminhos que levam até o estado 4 que representa uma falha e um único caminho até o estado 11 que representa o estado final de uma execução bem sucedida gerando um total de 6 diferentes caminhos.

Nó	Meta	Papel	Descrição
1	$start$	$bool$	Inicio do Teste
2	wtitle	*editor*	Um *editor* escreve o título do artigo.
3	wabs	*editor*	Um *editor* escreve o resumo.
5	wsectitle	*editor*	Um *editor* escreve o título das seções.
6	fds	$bool$	primeira versão do rascunho foi escrita.
7	wsec	*writer*	Um *writer* escreve as seções.
8	wcon, wref	*editor*, *writer*	Um *editor* escreve a conclusão enquanto um *writer* escreve as referências.
9	finish	$bool$	conclusão e referências foram escritas
10	sv	$bool$	segunda verão escrita
11	wp	$bool$	O artigo foi escrito

Tabela 8: Exemplo de caso de teste para *Write Paper* sem falha

5.1.2.6 Geração dos Casos de Teste

Agora que temos o número total de caminhos de teste calculado através de um diagrama de controle de fluxo do espaço de estados da RPC, precisamos gerar uma descrição para cada um desses caminhos que serão nossos casos de teste. Vamos primeiramente separar dois caminhos como exemplo, no primeiro a execução é bem sucedida e no segundo é onde ocorre uma falha:

1. $1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow 10 \rightarrow 11$.

2. $1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 4$.

Vamos começar pelo primeiro caminho, usando a Tabela 7 para identificar o lugar e a ficha que pertencem a determinado nó, identificamos as metas e os papéis de nosso caso de teste. Já na Tabela 6 vamos extrair a descrição de cada meta. Assim geramos a Tabela 8 com nosso primeiro caso de teste.

O segundo caminho que temos como exemplo é um caminho de falha, novamente usaremos as Tabelas 7 e 6 para gerar a Tabela 9 com o segundo caso de teste. Nesse exemplo, ao executar a meta $wsectitle$ uma falha ocorre impedindo a progressão para fds e indo para o lugar denominado $fail$.

Nó	Meta	Papel	Descrição
1	$start$	$bool$	Inicio do Teste
2	wtitle	*editor*	Um *editor* escreve o título do artigo.
3	wabs	*editor*	Um *editor* escreve o resumo.
5	wsectitle	*editor*	Um *editor* escreve o título das seções.
4	$fail$	$bool$	Uma falha ocorreu.

Tabela 9: Exemplo de caso de teste para *Write Paper* com falha

5.2 Soccer Team

5.2.1 Descrição do Cenário

A especificação estrutural do *Soccer Team* pode ser visualizada na Figura 25 onde podemos identificar os papéis *goalkeeper* (goleiro) e *back* (zagueiro) que pertencem ao grupo *defense* (defesa), *middle* (meia) e *attacker* (atacante) do grupo *attack* (ataque) e o papel *coach* (treinador). Nesse cenário a meta é marcar um gol, e para tal usaremos apenas dos papéis *goalkeeper*, *back*, *middle* e *attacker* que são sub-papéis de *player* (jogador).

A especificação funcional que irá coordenar os grupos de agentes para atingir suas metas é definido na Figura 26. Aqui, uma árvore de decomposição de metas é apresentada e a leitura é feita da seguinte maneira:

1. A meta $g6$ indica que alguém da equipe está com a bola no campo de defesa.
2. É realizada uma escolha entre as metas $g7$ (passar a bola para a meia esquerda do campo) e $g8$ (passar para a meia direita).
3. É feita outra escolha entre $g16$ e $g17$ para definir o lado que a bola foi recebida no meio campo.
4. As metas $g18$ que tem como função posicionar um atacante para receber a bola e $g19$ onde um meia se prepara para passar a bola para um atacante, são executadas em paralelo.
5. A meta $g14$ onde um meia passa a bola para um atacante é realizada.
6. Por fim um atacante realiza uma escolha entre $g20$ (chutar a bola no lado esquerdo do gol) e $g21$ (chutar a bola no lado direito do gol).

papel	relação deôntica	missão
goalkeeper	permissão	$m1$
back	permissão	$m1$
middle	obrigação	$m2$
attacker	obrigação	$m3$

Tabela 10: Especificação Deôntica para o cenário *Soccer Team*

A Tabela 10 representa a especificação deôntica para esse cenário e define as permissões e obrigações dos papéis que assumem as missões. As missões são definidas por: $m1$ referente a transferir a bola da defesa para o meio campo, $m2$ referente a transferir a bola do meio para o ataque, $m3$ é referente a posicionar os atacantes e chutar a bola a gol. Podemos notar na ED que ambos os papéis *goalkeeper* e *back* posssuem permissão para se comprometer a cumprir a missão $m1$, porém apenas um agente que pertença a um desses papéis irá assumir essa missão. A descrição de todas as metas para este exemplo pode ser vista na Tabela 11.

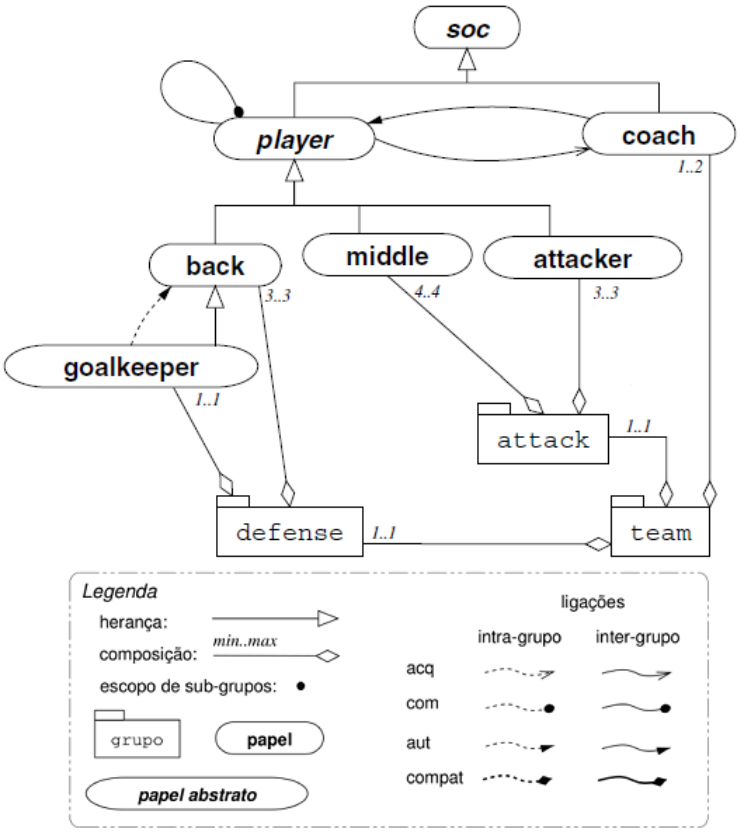


Figura 25: Especificação Estrutural *Soccer Team*, adaptado de (HÜBNER; SICHMAN; BOISSIER, 2002)

meta	descrição
g1	um gol foi marcado
g2	a bola está no meio campo
g3	a bola está no campo de ataque
g4	a bola foi chutada para o gol
g6	um jogador de defesa recebeu a bola
g7	a bola foi passada para a meia esquerda
g8	a bola foi passada para a meia direita
g9	a bola foi passada para o meio campo
g11	um meia está com a bola
g13	um atacante está numa boa posição
g14	um meia passa a bola para um atacante
g16	um meia recebe a bola no lado esquerdo
g17	um meia recebe a bola no lado direito
g18	um atacante se posiciona para receber a bola
g19	um meia se prepara para passar a bola
g20	um atacante chuta a bola no lado esquerdo do gol
g21	um atacante chuta a bola no lado direito do gol

Tabela 11: Descrição das metas *Soccer Team* (HÜBNER; SICHMAN; BOISSIER, 2002)

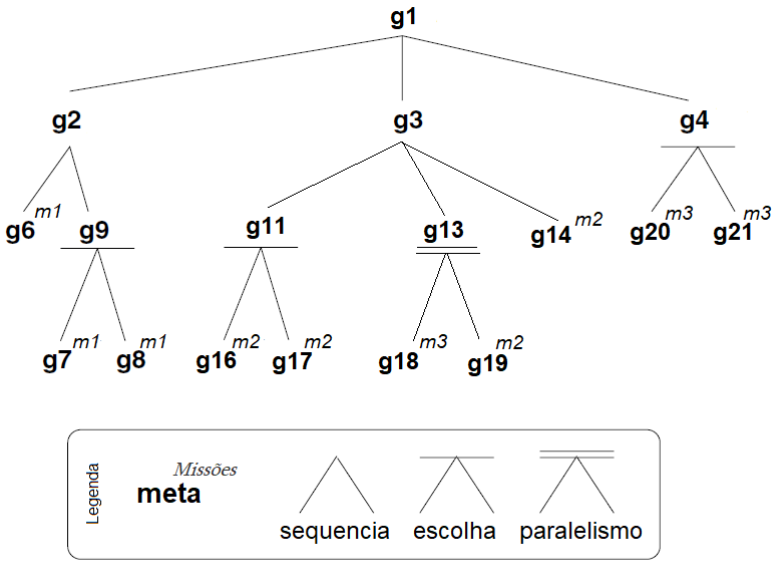


Figura 26: Especificação Funcional *Soccer Team*, adaptado de (HÜBNER; SICHMAN; BOISSIER, 2002)

5.2.2 Implementação

Feito a descrição do cenário a próxima fase é a implementação do método seguindo a sequência de etapas estabelecida:

5.2.2.1 Inclusão das Declarações

Extraindo os papéis da especificação estrutural utilizaremos o $goalkeeper$ (goleiro), $back$ (defensor), $middle$ (meia) e $attacker$ (atacante) que farão parte do conjunto de cores $PLAYER$, juntamente com as variáveis que serão utilizadas para efetuar as transições da rede:

```
colset PAPEL = with goalkeeper | back | attacker | middle ;
var p,p1,p2: PAPEL;
```

5.2.2.2 Geração da Estrutura

Com as declarações feitas deve-se gerar uma estrutura da rede utilizando os operadores de planos juntamente com a árvore de decomposição de metas da Figura 26. Na Figura 27 é possível visualizar a estrutura da rede, onde podemos identificar três escolhas e um paralelismo o que confere com a EF da Figura 26.

5.2.2.3 Inclusão das Inscrições

Essa etapa constituí na inclusão das inscrições na rede conforme Figura 30 onde os lugares representam as metas e as fichas identificam os papéis. A inclusão das inscrições corresponde a: colocar as expressões de arco para mudança de papéis; variáveis dos arcos;

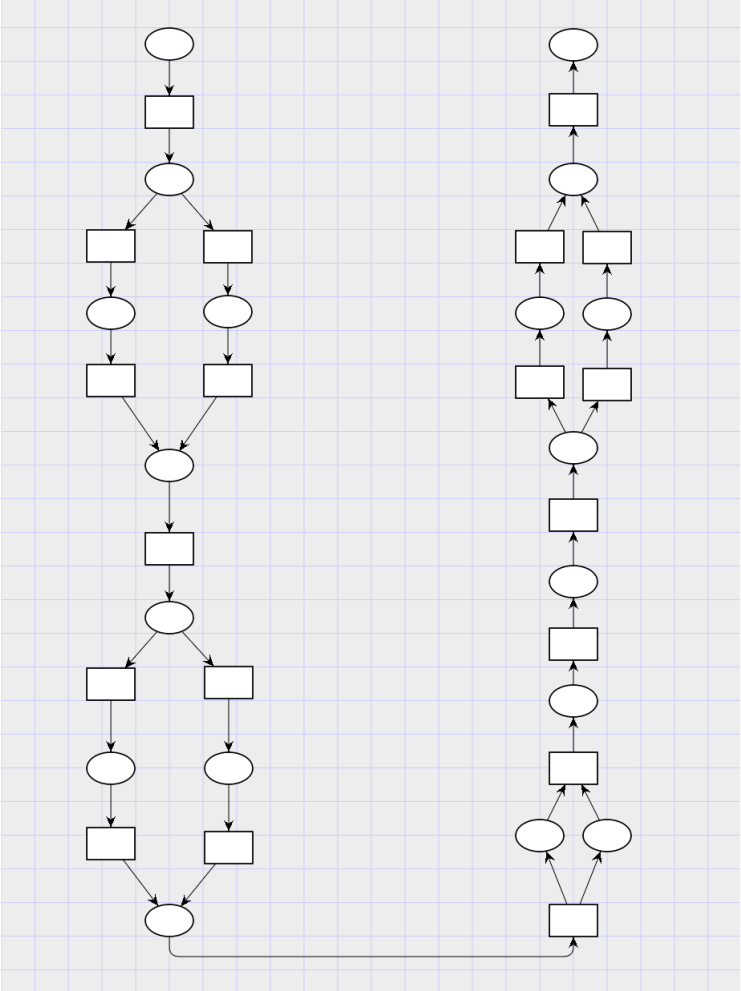


Figura 27: Estrura da RPCpM.

identificação dos lugares com nome da meta e o tipo de cor, por exemplo a meta $g6$ com o tipo $PAPEL$; além de uma identificação diferente para cada transição.

É interessante destacar que após a ficha passar pela primeira transição $t1$ a escolha entre um dos papéis, $back$ ou $goalkeeper$, para realização da meta $g6$ é feita através de uma função de expressão de arco que define um deles de forma randômica. O papel escolhido será responsável por satisfazer a meta $g6$ e realizar uma escolha entre as metas $g7$ e $g8$. Ao satisfazer uma dessas duas metas, a meta $g9$ automaticamente será dada como satisfeita, o que faz com que $g9$ não tenha nenhuma missão vinculada como é possível ver na Figura 26. Assim $g9$ e as demais metas semelhante a ela recebem na rede um lugar do tipo $BOOL$.

5.2.2.4 Inclusão das Falhas

Para a inclusão dos caminhos de falha serão considerados todos os lugares da rede que não sejam do tipo $BOOL$, ou seja, sobrando apenas as metas que possuem uma missão vinculada e portanto um papel para realizar ações que satisfaçam essa meta. Nesses lugares, um caminho de falha será criado para um mesmo estado, indicando que uma falha

ocorreu durante a execução do programa. Nesse exemplo conforme podemos visualizar na Figura 31 existem nove diferentes caminhos de falha.

A inclusão das falhas foi a última etapa de mapeamento em uma RPC baseada numa especificação $\mathcal{M}$oise^{+} ou RPCpM para o exemplo *Soccer Team*. Agora a rede se encontra concluída e já é possível rodar simulações nela.

5.2.2.5 Contagem dos Casos de Teste

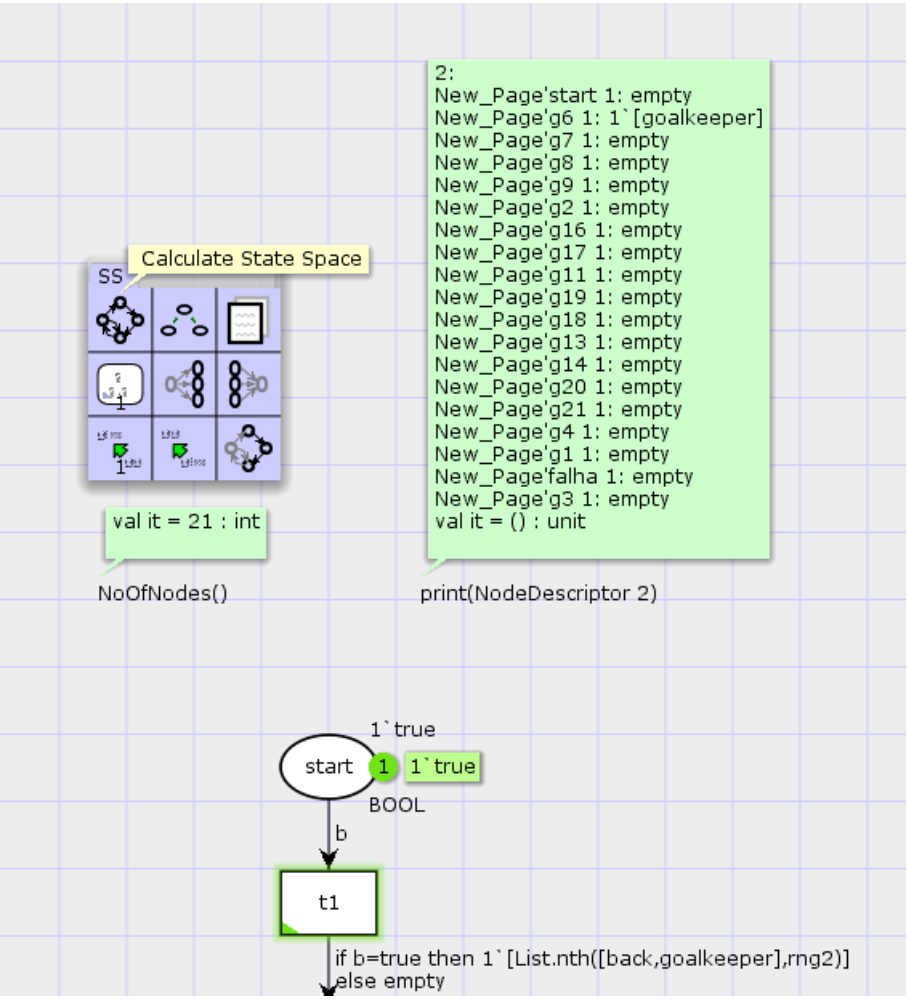


Figura 28: Contagem dos espaços de estados para o exemplo *Soccer Team*.

Nessa etapa serão contabilizados os caminhos de teste do sistema utilizando o critério de adequação **caminho de transição dos estados**. O primeiro passo será a identificação de todos os nós de estado que compõem os espaços de estado da rede. Para isso será utilizado as funções de espaço de estado da ferramenta CPNTools.

Na Figura 28 é possível ver a descrição para o nó de estado número 2 da rede *Soccer Team* onde a ficha $goalkeeper$ se encontra no lugar $g6$ e todos os outros lugares da rede se encontram vazios (*empty*). Para visualizar todos os possíveis estados dessa rede será necessário utilizar a função $print(NodeDescriptor\ n)$ para todos os 21 nós do espaço de

estado de nosso exemplo. Na Tabela 12 temos as marcações para cada estado ignorando os lugares vazios e mostrando apenas lugares com fichas.

Estado	Lugar	Ficha	Estado	Lugar	Ficha
1	start	true	12	g16	middle
2	g6	goalkeeper	13	g11	true
3	g6	back	14	g19, g18	middle, attacker
4	falha	falha	15	g13	true
5	g8	goalkeeper	16	g14	middle
6	g7	goalkeeper	17	g3	true
7	g8	back	18	g21	attacker
8	g7	back	19	g20	attacker
9	g9	true	20	g4	true
10	g2	true	21	g1	true
11	g17	middle			

Tabela 12: Espaços de Estado do *Soccer Team*

Agora que foi gerada uma tabela com todos os possíveis estados da RPC é gerado um grafo de controle de fluxo que mostra a ligação entre esses estados. A geração desse grafo é importante para que possamos visualmente identificar todos os diferentes caminhos, já que, a ferramenta CPNTools não disponibiliza um relatório que gere os caminhos dos nós do espaço de estados.

Para gerar esse grafo devemos observar a Tabela 12 começando pelo primeiro estado 1 que possui o lugar $start$ e através da RPC da Figura 31 analisar para quais lugares esse estado leva, no caso é $g6$. Novamente na Tabela 12 veremos quais estados possuem o lugar $g6$, que no caso são o 2 e o 3 e fazer uma ligação no grafo saindo do nó 1 para os nós 2 e 3. Repetimos esse processo para todos os outros nós até que o grafo seja completado. Na Figura 29 se encontra o grafo de controle de fluxo completo para todos os nós que representam os 21 estados, do espaço de estados para o exemplo *Soccer Team*.

Gerado o grafo de controle de fluxo, podemos realizar a contagem dos caminhos de estado de nossa rede, o que é possível utilizando a Definição 5 para identificação de cada caminho do grafo. Em nosso grafo da Figura 29 o nó inicial v_o é o nó 1 e, temos dois nós finais v_n que são os nós 4, representando a falha, e 21 que é o estado final para uma execução bem sucedida. Portanto o número de caminhos que corresponde todas as possíveis execuções do sistema deve englobar o somatório de todos caminhos do nó 1 ao nó 4 assim como do nó 1 ao nó 21. Por se tratar de um grafo de tamanho grande e com diferentes possibilidades de caminhos, um algoritmo[1] com a técnica recursiva *backtracking* foi utilizado para realizar o somatório dos caminhos, tendo sido contabilizados um total de 62 diferentes caminhos sendo que 46 deles correspondem aos caminhos que terminam no nó 4 e 16 no nó 21.

[1]https://www.geeksforgeeks.org/count-possible-paths-two-vertices/

5.2.2.6 Geração dos Casos de Teste

Após a identificação e contagem dos casos de teste a ultima etapa de nossa metodologia é a descrição detalhada para cada um deles. Como foi encontrado um número de 62 caminhos, será inviável apresentar todos, portanto mostraremos dois exemplos desses casos de teste, um para quando a execução é bem sucedida e outro quando ela falha. Primeiramente vamos identificar quais são esses caminhos em nosso grafo da Figura 29, sendo eles:

1. $1 \rightarrow 3 \rightarrow 8 \rightarrow 9 \rightarrow 10 \rightarrow 12 \rightarrow 13 \rightarrow 14 \rightarrow 15 \rightarrow 16 \rightarrow 17 \rightarrow 18 \rightarrow 20 \rightarrow 21.$
2. $1 \rightarrow 3 \rightarrow 8 \rightarrow 4.$

Utilizando a Tabela 12 para identificar qual lugar e Ficha da RPC pertence cada nó, identificando as metas e os papéis de nosso caso de teste, juntamente com a Tabela 11 que fornecerá uma descrição para essas metas, é gerada a Tabela 13 com uma descrição do caso de teste referente ao caminho bem sucedido escolhido.

Nó	**Meta**	**Papel**	**Descrição**
1	$start$	bool	Inicia o Teste
3	g6	*back*	Um *back* recebe a bola no campo de defesa.
8	g7	*back*	Um *back* passa a bola para a meia esquerda.
9	g9	bool	A bola foi passada para o meio campo
10	g2	bool	A bola está no meio campo
12	g16	*middle*	Um *middle* recebeu a bola no lado esquerdo
13	g11	bool	Um *middle* está com a bola
14	g19, g18	*middle*, *attacker*	um *middle* se prepara para passar a bola e um *attacker* se posiciona para receber a bola
15	g13	bool	Um *attacker* está numa boa posição
16	g14	*middle*	Um *middle* passa a bola para um atacante
17	g3	bool	A bola está no campo de ataque
18	g21	*attacker*	Um *attacker* chuta no lado direito do gol
20	g4	bool	A bola foi chutada para o gol
21	g1	bool	Um gol foi marcado

Tabela 13: Exemplo de caso de teste para *Soccer Team* sem falha

Vamos fazer o mesmo procedimento para o exemplo 2 onde uma falha na execução da rede acontece. A ideia de um caminho de falha não é necessariamente para simular essa falha, e sim prever essa possibilidade para o caso dela acontecer durante a execução dos testes. Na Tabela 14 está a descrição para esse caso de teste.

Nó	Meta	Papel	Descrição
1	$start$	bool	Inicia o Teste
3	g6	*back*	Um *back* recebe a bola no campo de defesa.
8	g7	*back*	Um *back* passa a bola para a meia esquerda.
4	falha	bool	Uma falha ocorre na execução da meta g7

Tabela 14: Exemplo de caso de teste para *Soccer Team* com falha

5.2.3 Teste do Sistema

Com o objetivo de apresentar uma utilização prática do método, o cenário acima descrito, *Soccer Team*[2], foi implementado utilizando o *framework* JaCaMo[3] (BOISSIER et al., 2013), uma plataforma de desenvolvimento para SMA com suporte ao $\mathcal{M}$oise$^+$, na expectativa de gerar execuções de teste baseadas nos casos de teste identificados pelo método. Na Figura 32 podemos ver um trecho do código fonte que corresponde a EF apresentada na Figura 26 .

No simulador existem quatro agentes cada um com um respectivo papel, sendo eles: mateus com o papel $back$, roberto no papel de $goalkeeper$, lucas $middle$ e jonny um $attacker$.

Utilizando o caso de teste gerado na Tabela 13 como guia para a execução do teste, foi realizada uma simulação na qual podemos ver na Figura 33. Nessa execução o agente mateus após anunciar que esta de posse da bola (nó 3), passa a bola para o meio campo esquerdo (nó 8). O agente lucas por sua vez indica que recebeu a bola no lado esquerdo (nó 12) e anuncia que está decidindo para onde passar a bola enquanto o atacante jonny se posiciona para receber o passe (nó 14). Após essa etapa lucas passa a bola para jonny (nó 16), e jonny após receber a bola chuta no lado direito do gol (nó 18).

O teste foi bem sucedido embora tenha sido um modelo simplificado para demonstrar a utilização do método num sistema real. Uma implementação mais complexa com interação entre os agentes e um ambiente, além de uma interface gráfica, podem ser futuramente desenvolvidos para que possam ser realizados testes mais realistas.

[2]https://github.com/ricardoarend/SoccerTeam
[3]http://jacamo.sourceforge.net/

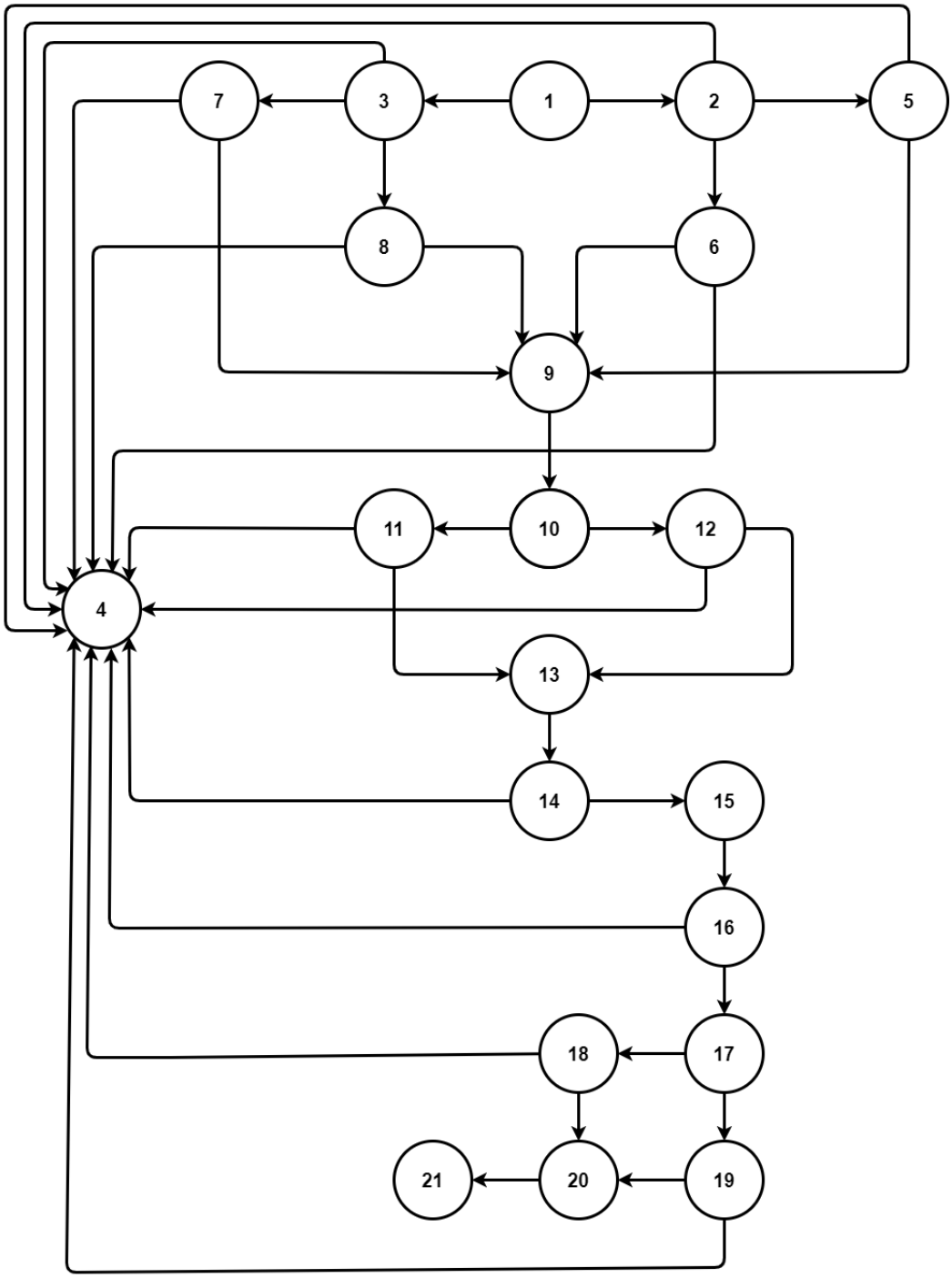


Figura 29: Diagrama do Espaço de Estados do *Soccer Team*.

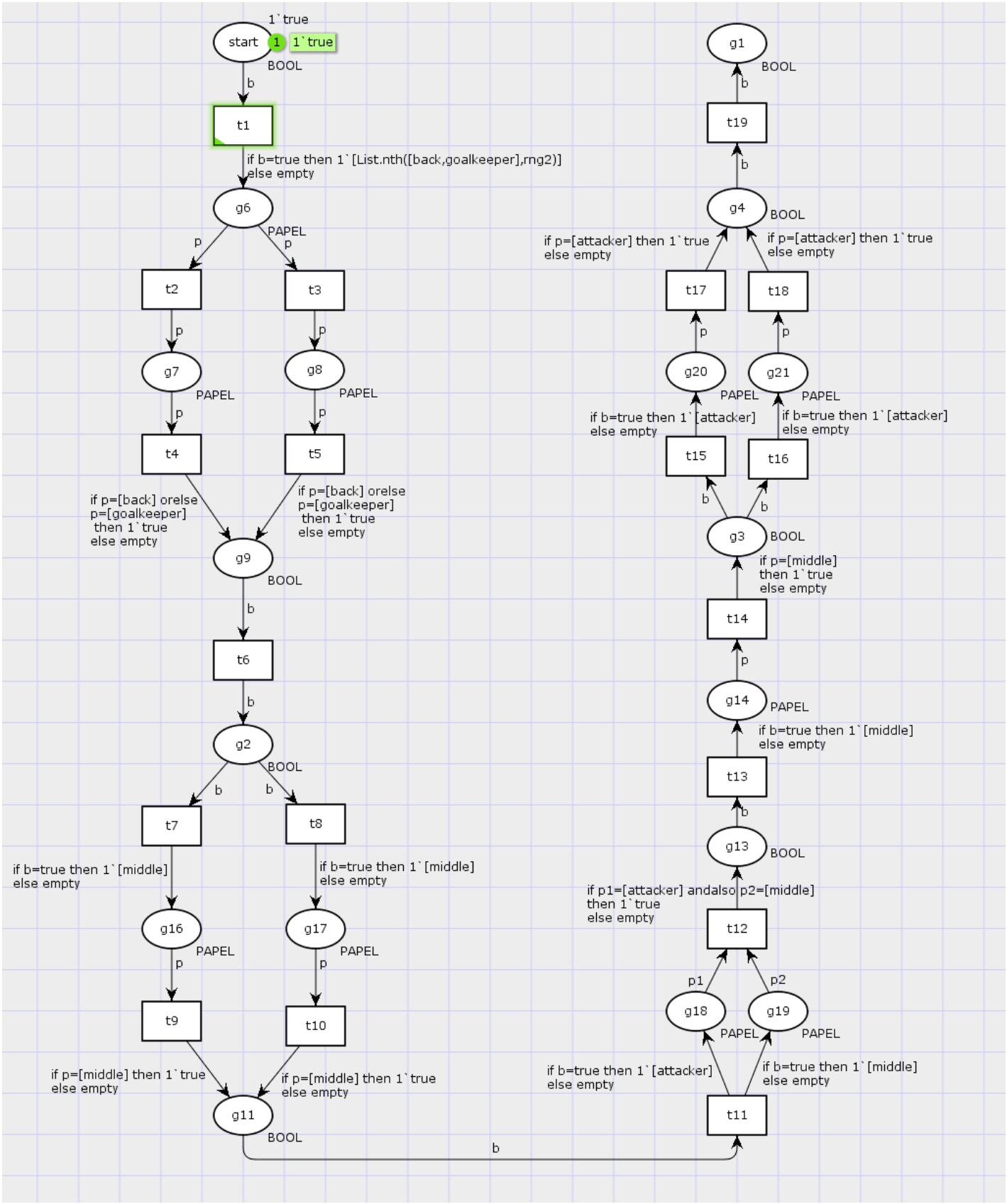


Figura 30: *Soccer Team* no formato RPCpM com as inscrições.

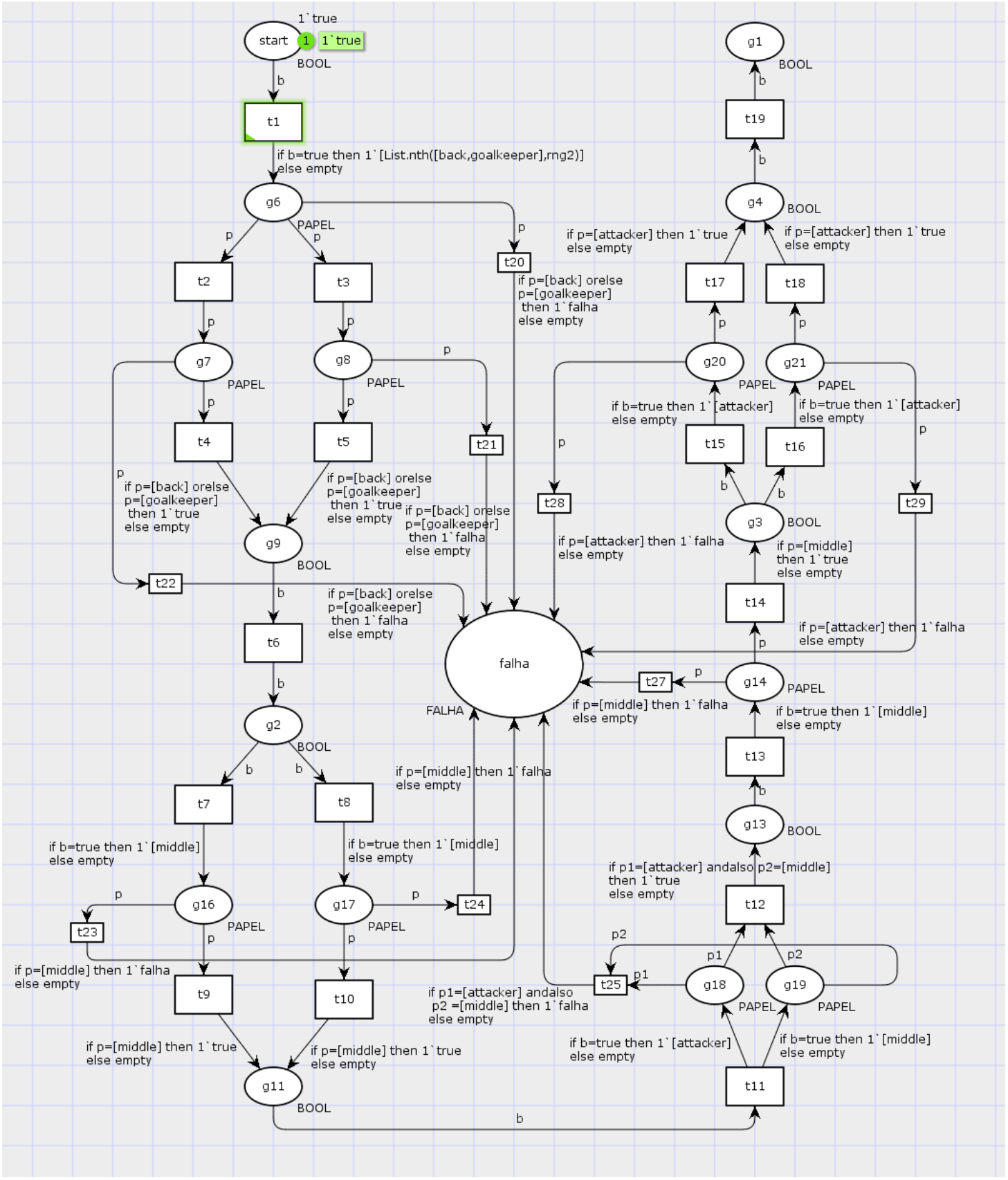


Figura 31: *Soccer Team* no formato RPC*p*M em sua versão final com caminhos de falha.

```
42
43
44    <functional-specification>
45        <scheme id="marcarGol" >
46
47            <goal id="g1">
48                <plan operator="sequence" >
49                    <goal id="g2" >
50                        <plan operator="sequence">
51                            <goal id="g6"/>
52                            <goal id="g9">
53                                <plan operator="choice" >
54                                    <goal id="g7"/>
55                                    <goal id="g8"/>
56                                </plan>
57                            </goal>
58                        </plan>
59                    </goal>
60                    <goal id="g3">
61                        <plan operator="sequence">
62                            <goal id="g11">
63                                <plan operator="choice" >
64                                    <goal id="g16"/>
65                                    <goal id="g17"/>
66                                </plan>
67                            </goal>
68                            <goal id="g13">
69                                <plan operator="parallel" >
70                                    <goal id="g18"/>
71                                    <goal id="g19"/>
72                                </plan>
73                            </goal>
74                            <goal id="g14"/>
75                        </plan>
76                    </goal>
77                    <goal id="g4" >
78                        <plan operator="choice" >
79                            <goal id="g20"/>
80                            <goal id="g21"/>
81                        </plan>
82                    </goal>
83                </plan>
84            </goal>
85
86            <mission id="m1" min="1" max="1">
87                <goal id="g6"/>
88                <goal id="g7"/>
89                <goal id="g8"/>
90            </mission>
91
92             <mission id="m2" min="1" max="1">
93                <goal id="g14"/>
94                <goal id="g16"/>
95                <goal id="g17"/>
96                <goal id="g19"/>
97            </mission>
98
99            <mission id="m3" min="1" max="1">
100               <goal id="g18"/>
101               <goal id="g20"/>
102               <goal id="g21"/>
103           </mission>
```

Figura 32: Código fonte da EF para o exemplo *Soccer Team*.

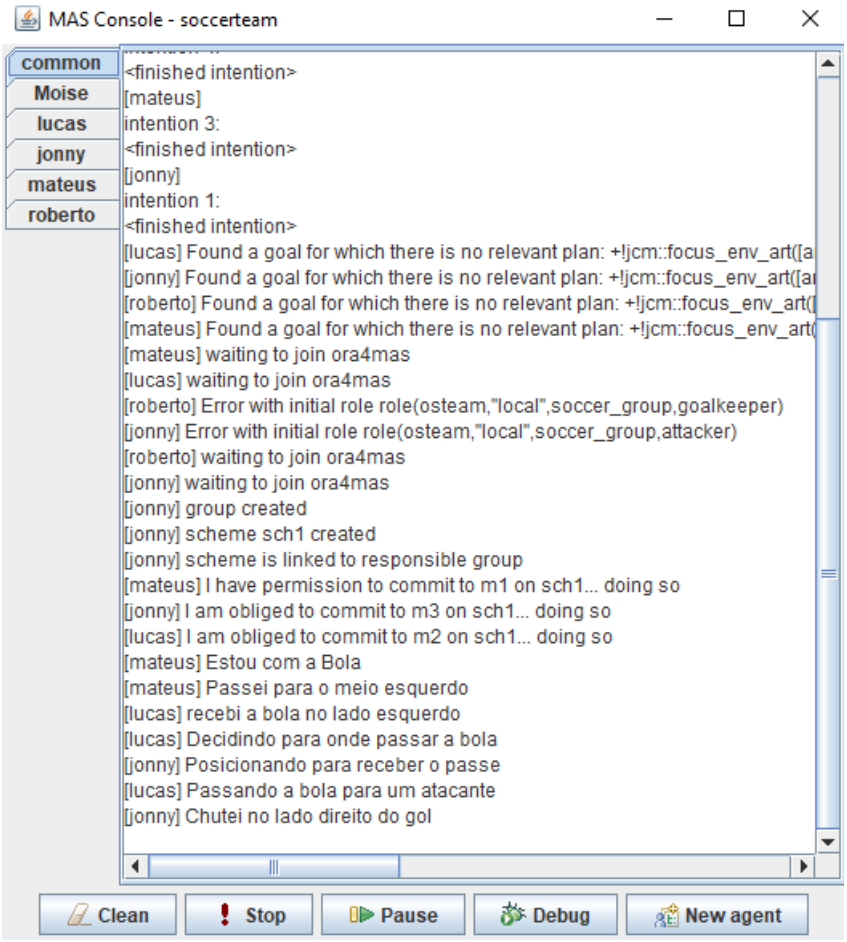


Figura 33: Console de execução do teste para o exemplo *Soccer Team*.

6 CONCLUSÃO

Como vimos, desenvolver mecanismos de teste para SMA é uma tarefa desafiadora pois se tratam de sistemas autônomos com diferentes possibilidades de saída para uma mesma entrada. Porém, se torna cada vez mais necessário a elaboração de mecanismos que contornem esses problemas buscando uma solução para a realização desses testes com eficiência.

Este trabalho de dissertação apresentou um método de testes para o modelo organizacional $\mathcal{M}$oise$^+$. Ele é constituído por: i) um método formal para avaliação da testabilidade do modelo em analise; ii) uma sistemática para a geração de casos de teste. Com esse método é possível quantificar e identificar os casos de teste necessários para a realização de testes no sistema.

O método anterior, desenvolvido por Rodrigues (2018), permitiu mapear em RPC sistemas multiagente modelados pelo $\mathcal{M}$oise$^+$ através de cinco etapas com os processos: declarações, estrutura, inscrições, falha e caminhos. O trabalho aqui apresentado veio dar uma continuidade a esse método, com a intenção de aprimorar alguns aspectos, como por exemplo uma revisão nas ocorrências de falhas, uma devida formalização do método, a utilização de um critério de adequação de teste mais adequado e um exemplo de caso de uso com maior complexidade contendo o operador de escolha e diferentes papéis para uma mesma meta. Além dessas mudanças também foi acrescentado um método para a geração dos casos de teste.

Podemos então dizer que a principal contribuição deste trabalho foi a elaboração de um método para geração e especificação dos casos de testes. Porém, além dela existem outras contribuições secundárias como: i) A definição formal para um mapeamento de Rede de Petri colorida para $\mathcal{M}$oise$^+$; ii) Uma nova abordagem para a avaliação da testabilidade do sistema utilizando o critério de adequação **caminho de transição dos estados**.

Nos exemplos de caso de uso apresentados foi possível aplicar o método de maneira clara e objetiva. Os resultados obtidos foram satisfatórios com o processo iniciando na mapeamento da rede, passando pela contagem dos caminhos de teste e por fim a geração dos casos de teste. Em ambos os exemplos todas essas etapas foram concluídas, demonstrando a eficácia e robustez do método. Além disso foi desenvolvido uma aplicação para

simular um desses exemplos, chamado de *Soccer Team*, onde os casos de teste gerados foram utilizados como guia para execuções de testes no sistema.

6.1 Trabalhos Futuros

Como sugestões de trabalhos futuros podemos citar:

- O desenvolvimento de uma ferramenta para gerar o grafo de controle de fluxo referente aos espaços de estados da RPC com a possibilidade de contar os caminhos e gerar as tabelas de caso de teste apenas com a inclusão dos dados essenciais pelo usuário.
- Aprimorar os exemplos para teste com SMA complexos onde haja interferência do ambiente e maiores possibilidades da ocorrência de falhas.
- Realizar um estudo sobre diferentes tipos de especificações $\mathcal{M}$oise$^+$ e identificar qual o melhor método de teste para cada tipo de especificação.

7 TRABALHOS PUBLICADOS

A partir das contribuições deste projeto, até o momento foram publicados dois trabalhos, a saber:

1. MACHADO, R. A.; GONÇALVES, E. M. Uma revisão dos métodos para geração e execução de testes em sistemas multiagentes. WESAAC, p. 25–35, 2019.

2. GONÇALVES, E. M.; RODRIGUES, B. C.; MACHADO, R. A. Assessment of Testability on Multiagent SystemsDeveloped with Organizational Model $\mathcal{M}$oise. EPIA Conference on Artificial Intelligence, 2019.

REFERÊNCIAS

ABDULLAH, D.; BURNSTEIN, I. *Practical Software Testing*. [S.l.]: Springer, 2003.

ABRAN, A. et al. Software engineering body of knowledge. *IEEE Computer Society, Angela Burgess*, 2004.

ADRION, W. R.; BRANSTAD, M. A.; CHERNIAVSKY, J. C. Validation, verification, and testing of computer software. *ACM Computing Surveys (CSUR)*, ACM, v. 14, n. 2, p. 159–192, 1982.

AMMANN, P.; OFFUTT, J. *Introduction to software testing*. [S.l.]: Cambridge University Press, 2016.

ARGENTE, E.; JULIAN, V.; BOTTI, V. Multi-agent system development based on organizations. *Electronic Notes in Theoretical Computer Science*, Elsevier, v. 150, n. 3, p. 55–71, 2006.

BARNIER, C. et al. Toward an embedded multi-agent system methodology and positioning on testing. In: IEEE. *2017 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. [S.l.], 2017. p. 239–244.

BOISSIER, O. et al. Multi-agent oriented programming with jacamo. *Science of Computer Programming*, Elsevier, v. 78, n. 6, p. 747–761, 2013.

BORDINI, R. H.; HÜBNER, J. F.; WOOLDRIDGE, M. *Programming multi-agent systems in AgentSpeak using Jason*. [S.l.]: John Wiley & Sons, 2007. v. 8.

BRATMAN, M. *Intention, plans, and practical reason*. [S.l.]: Harvard University Press Cambridge, MA, 1987. v. 10.

BRAUBACH, L.; LAMERSDORF, W.; POKAHR, A. Jadex: Implementing a bdi-infrastructure for jade agents. Citeseer, 2003.

CARDOSO, J.; VALETTE, R. *Petri Nets. Original title: Redes de Petri*. [S.l.]: Editora da UFSC, 1997. v. 1.

CLAVEL, M. et al. Maude: Specification and programming in rewriting logic. *Theoretical Computer Science*, Elsevier, v. 285, n. 2, p. 187–243, 2002.

COELHO, R. et al. Jat: A test automation framework for multi-agent systems. In: IEEE. *Software Maintenance, 2007. ICSM 2007. IEEE International Conference on*. [S.l.], 2007. p. 425–434.

DEMAZEAU, Y.; MÜLLER, J.-P. *Decentralized Ai*. [S.l.]: Elsevier, 1990. v. 2.

DIGNUM, V. *Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models: Semantics and Dynamics of Organizational Models*. [S.l.]: IGI Global, 2009.

EASSA, F. E. et al. Dttas: A dynamic testing tool for agent-based systems. *Pensee Journal*, v. 76, n. 5, 2014.

FERBER, J.; GASSER, L. Intelligence artificielle distribuée.(1991). *Tutorial Notes of the 11 thConference on Expert Systems and their Applications Avignon, France*, 1991.

FERBER, J.; GUTKNECHT, O.; MICHEL, F. From agents to organizations: an organizational view of multi-agent systems. In: SPRINGER. *International workshop on agent-oriented software engineering*. [S.l.], 2003. p. 214–230.

GIRAULT, C.; VALK, R. *Petri nets for systems engineering: a guide to modelling, verification, and applications*. [S.l.]: Springer Science & Business Media, 2013.

GIUNCHIGLIA, F.; MYLOPOULOS, J.; PERINI, A. The tropos software development methodology: processes, models and diagrams. In: SPRINGER. *International Workshop on Agent-Oriented Software Engineering*. [S.l.], 2002. p. 162–173.

GOKULAN, B. P.; SRINIVASAN, D. An introduction to multi-agent systems. In: ____. [S.l.: s.n.], 2010. v. 310, p. 1–27.

GOMEZ-SANZ, J. J. et al. Testing and debugging of mas interactions with ingenias. In: SPRINGER. *International Workshop on Agent-Oriented Software Engineering*. [S.l.], 2008. p. 199–212.

GONÇALVES, E. M.; QUADROS, C.; SALDANHA, J. Uma análise comparativa da especificaçao formal em sistemas multi-agente: os desafios e as exigências uma década depois. In: *Anais do X Workshop-Escola de Sistemas de Agentes, seus Ambientes e Aplicaçoes*. Maceio, Brazil: [s.n.], 2016. v. 1.

GONÇALVES, E. M.; RODRIGUES, B. C.; MACHADO, R. A. Assessment of testability on multiagent systems developed with organizational model $\mathcal{M}$oise. In: SPRINGER. *EPIA Conference on Artificial Intelligence*. [S.l.], 2019. p. 581–592.

HANNOUN, M. et al. Moise: An organizational model for multi-agent systems. In: *Advances in Artificial Intelligence*. [S.l.]: Springer, 2000. p. 156–165.

HOUHAMDI, Z. Multi-agent system testing: A survey. *International Journal of Advanced Computer*, Citeseer, 2011.

HOUHAMDI, Z.; ATHAMENA, B. Structured system test suite generation process for multi-agent system. *International Journal on Computer Science and Engineering*, Citeseer, v. 3, n. 4, p. 1681–1688, 2011.

HUBER, M. J. Jam: A bdi-theoretic mobile agent architecture. In: ACM. *Proceedings of the third annual conference on Autonomous Agents*. [S.l.], 1999. p. 236–243.

HUBNER, J.; SICHMAN, J.; BOISSIER, O. *Using the Moise+ for a Cooperative Framework of MAS Reorganisation, SBIA 04*. [S.l.]: Springer, 2004.

HÜBNER, J. F. *Um modelo de reorganização de sistemas multiagentes.* Tese (Doutorado) — Universidade de São Paulo, 2003.

HÜBNER, J. F.; BOISSIER, O.; BORDINI, R. H. A normative programming language for multi-agent organisations. *Annals of Mathematics and Artificial Intelligence*, Springer, v. 62, n. 1-2, p. 27–53, 2011.

HÜBNER, J. F.; SICHMAN, J. S.; BOISSIER, O. A model for the structural, functional, and deontic specification of organizations in multiagent systems. In: SPRINGER. *Brazilian Symposium on Artificial Intelligence*. [S.l.], 2002. p. 118–128.

HÜBNER, J. F.; SICHMAN, J. S.; BOISSIER, O. S-moise+: A middleware for developing organised multi-agent systems. In: SPRINGER. *International Conference on Autonomous Agents and Multiagent Systems*. [S.l.], 2005. p. 64–77.

IEEE Guide for Software Verification and Validation Plans. [S.l.], 1994. 1-87 p.

IEEE Standard Glossary of Software Engineering Terminology. [S.l.], 1990. 1-84 p.

JENSEN, K. *Coloured Petri nets: basic concepts, analysis methods and practical use*. [S.l.]: Springer Science & Business Media, 1997. v. 1.

JENSEN, K.; KRISTENSEN, L. M. *Coloured Petri nets: modelling and validation of concurrent systems*. [S.l.]: Springer Science & Business Media, 2009.

KERRAOUI, S. et al. Matt: Multi agents testing tool based nets within nets. *Journal of Information and Organizational Sciences*, Fakultet organizacije i informatike Sveučilišta u Zagrebu, v. 40, n. 2, p. 165–184, 2016.

KISSOUM, Y.; SAHNOUN, Z. Test cases generation for multi-agent systems using formal specification. *Computer Systems and Applications*, p. 76–83, 2007.

LEMAÎTRE, C.; EXCELENTE, C.; FALLAH-SEGHROUCHNI, A. E. Multi-agent organization approach to electronic business automation. In: *5th International Conference on Decision Science Institute, Athènes*. [S.l.: s.n.], 1999.

MACIEL, P. R.; LINS, R. D.; CUNHA, P. R. *Introdução às redes de Petri e aplicações*. [S.l.]: UNICAMP-Instituto de Computacao, 1996.

MATHUR, A. P. *Foundations of software testing, 2/e*. [S.l.]: Pearson Education India, 2013.

MILLER, T.; PADGHAM, L.; THANGARAJAH, J. Test coverage criteria for agent interaction testing. In: SPRINGER. *International Workshop on Agent-Oriented Software Engineering*. [S.l.], 2010. p. 91–105.

MURATA, T. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, IEEE, v. 77, n. 4, p. 541–580, 1989.

NASCIMENTO, N. M. do et al. A publish-subscribe based architecture for testing multiagent systems. In: *SEKE*. [S.l.: s.n.], 2017. p. 521–526.

NGUYEN, C. D.; PERINI, A.; TONELLA, P. Ontology-based test generation for multiagent systems. In: INTERNATIONAL FOUNDATION FOR AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS. *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 3*. [S.l.], 2008. p. 1315–1320.

NGUYEN, C. D. et al. Automated continuous testing of multi-agent systems. In: CITESEER. *The fifth European workshop on Multi-agent systems*. [S.l.], 2007.

NGUYEN, D. C. *Testing techniques for software agents*. Tese (Doutorado) — University of Trento, 2009.

PADGHAM, L.; THANGARAJAH, J.; WINIKOFF, M. Prometheus design tool. In: AAAI PRESS. *AAAI 2008*. [S.l.], 2008.

PADGHAM, L.; WINIKOFF, M. *Developing intelligent agent systems: A practical guide*. [S.l.]: John Wiley & Sons, 2005. v. 13.

PAVÓN, J.; GÓMEZ-SANZ, J. J.; FUENTES, R. The ingenias methodology and tools. In: *Agent-oriented methodologies*. [S.l.]: IGI Global, 2005. p. 236–276.

POUTAKIDIS, D. et al. Debugging and testing of multi-agent systems using design artefacts. In: *Multi-Agent Programming:*. [S.l.]: Springer, 2009. p. 215–258.

RAO, A. S.; GEORGEFF, M. P. Modeling rational agents within a bdi-architecture. *KR*, v. 91, p. 473–484, 1991.

REHMAN, S. U.; NADEEM, A. An approach to model based testing of multiagent systems. *The Scientific World Journal*, Hindawi, v. 2015, 2015.

RICHARDSON, A. et al. Introduction to rabbitmq. *Google UK, available at http://www. rabbitmq. com/resources/google-tech-talk-final/alexis-google-rabbitmq-talk. pdf, retrieved on Mar*, v. 30, p. 33, 2012.

RODRIGUES, B. C. *Testabilidade de sistemas multiagentes organizados usando o modelo Moise: uma abordagem com Redes de Petri*. Dissertação (Mestrado) — Universidade Federal do Rio Grande, 2018.

RUSSELL, S.; NORVIG, P. Artificial intelligence: A modern approach prentice-hall. *Englewood cliffs, NJ*, 1995.

RUSSELL, S. J.; NORVIG, P. et al. *Artificial intelligence: a Modern Approach*. [S.l.]: Prentice Hall/Pearson Education,, 2003.

SALAMON, T. A three-layer approach to testing of multi-agent systems. In: *Information Systems Development*. [S.l.]: Springer, 2009. p. 393–401.

VLASSIS, N. A concise introduction to multiagent systems and distributed artificial intelligence. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, Morgan & Claypool Publishers, v. 1, n. 1, p. 1–71, 2007.

WANG, S.; ZHU, H. Catest: a test automation framework for multi-agent systems. In: IEEE. *Computer Software and Applications Conference (COMPSAC), 2012 IEEE 36th Annual*. [S.l.], 2012. p. 148–157.

WINIKOFF, M. Jack^{TM} intelligent agents: an industrial strength platform. In: *Multi-Agent Programming*. [S.l.]: Springer, 2005. p. 175–193.

WINIKOFF, M. Bdi agent testability revisited. *Autonomous Agents and Multi-Agent Systems*, Springer, v. 31, n. 5, p. 1094–1132, 2017.

WOOLDRIDGE, M. Intelligent agents. *Multiagent systems*, MIT Press London, v. 35, n. 4, p. 51, 1999.

WOOLDRIDGE, M.; JENNINGS, N. R. Intelligent agents: Theory and practice. *The knowledge engineering review*, Cambridge University Press, v. 10, n. 2, p. 115–152, 1995.

WOOLDRIDGE, M.; JENNINGS, N. R.; KINNY, D. The gaia methodology for agent-oriented analysis and design. *Autonomous Agents and multi-agent systems*, Springer, v. 3, n. 3, p. 285–312, 2000.

ZHANG, Z.; THANGARAJAH, J.; PADGHAM, L. Automated testing for intelligent agent systems. In: SPRINGER. *International Workshop on Agent-Oriented Software Engineering*. [S.l.], 2009. p. 66–79.

ZHU, H. A formal analysis of the subsume relation between software test adequacy criteria. *IEEE Transactions on Software Engineering*, IEEE, v. 22, n. 4, p. 248–255, 1996.

ZHU, H. Slabs: A formal specification language for agent-based systems. *International Journal of Software Engineering and Knowledge Engineering*, World Scientific, v. 11, n. 05, p. 529–558, 2001.

ZHU, H.; HALL, P. A.; MAY, J. H. Software unit test coverage and adequacy. *Acm computing surveys (csur)*, ACM, v. 29, n. 4, p. 366–427, 1997.

ZHU, H.; HE, X. A methodology of testing high-level petri nets. *Information and Software Technology*, Elsevier, v. 44, n. 8, p. 473–489, 2002.