Avaliação da testabilidade do modelo organizacional $\mathcal{M}$oise$^+$ baseada em Redes de Petri

Bruno Coelho Rodrigues, Eder Mateus Gonçalves

[1]Centro de Ciências Computacionais – Universidade Federal do Rio Grande (FURG)
Rio Grande – RS – Brasil

brunocoelho.r@gmail.com, edergoncalves@furg.br

Abstract. *Organizational models in Multiagent Systems (MAS) are used to structure agents into groups, where members have roles to play and also constraints to obey. Even with this level of control over agents, unexpected behaviors may arise. To ensure the correct functioning of the system, software testing techniques can be employed as one of the strategies. This work aims to propose a method to evaluate testability in MAS using the $\mathcal{M}$oise$^+$ organization model, using Petri Nets (PN) as a tool for description and analysis. The method is based on a testability evaluation technique for BDI agents, which should be mapped to Petri nets. The initial result indicates the number of use cases required to ensure system coverage.*

Resumo. *Modelos organizacionais em Sistemas Multiagentes (SMA) são empregados para estruturar os agentes em grupos, onde os membros têm papéis a desempenhar e também restrições a obedecer. Mesmo com este nível de controle sobre os agentes, comportamentos inesperados podem surgir. Para garantir o correto funcionamento do sistema, técnicas de teste de software podem ser empregadas como uma das estratégias. Este trabalho tem por objetivo propor um método para avaliar a testabilidade em SMA que emprega o modelo de organização $\mathcal{M}$oise$^+$, utilizando Rede de Petri (RP) como ferramenta de descrição e análise. O método é baseado em uma técnica de avaliação de testabilidade para agentes BDI, que deve ser mapeado para Redes de Petri. O resultado inicial indica o número de casos de uso necessários para garantir a cobertura do sistema.*

1. Introdução

Segundo [Hübner et al. 2007] em uma sociedade de agentes comportamentos indesejados podem emergir no sistema. Para resolver este tipo de comportamento, os SMA podem ser representados como uma organização através de modelos organizacionais. Estes modelos coordenam os agentes em grupos e hierarquias fazendo com que eles sigam regras comportamentais específicas [Van Den Broek et al. 2005, Argente et al. 2006].

Mesmo possuindo estruturas complexas de organização, um SMA ainda possui poucas garantias de que seu funcionamento é correto, sendo este um dos obstáculos para uma maior adoção desta abordagem na indústria [Houhamdi 2011, Winikoff 2010]. Um modo de obter essa garantia é através de Teste de Software (TS). Os testes ajudam a medir a qualidade do software em termos de números de defeitos encontra-

dos [Graham et al. 2008]. Mas já se sabe que testar SMA não é uma tarefa trivial [Winikoff 2010].

[Athamena and Houhamdi 2012] abordaram os testes de comportamento de SMA utilizando um modelo para transformar diagramas de classe de agente e função do MaSE para o diagrama UML 2.0 e então para RP equivalentes. [Winikoff and Cranefield 2014] avaliaram a dificuldade de testar um SMA BDI utilizando técnicas de caixa-branca baseado em todos os caminhos, neste trabalho o autor concluiu que tentar garantir a correção do sistema testando o sistema como um todo não é viável. Em [Winikoff 2017] o objetivo é o mesmo do trabalho anterior, mas agora utilizando como métrica todas as arestas, e não mais todos os caminhos. Com esta nova métrica, concluiu-se que o número de testes necessários é suficientemente pequeno para ser viável.

O objetivo deste trabalho é identificar o número de casos de teste necessários para a cobertura total de uma especificação $\mathcal{M}$oise$^+$, e isto é feito através da contagem de caminhos na RP. É utilizando como base o teste de caixa-branca, semelhante ao apresentado em [Winikoff 2017], a GPT utilizada para representar programas BDI é bastante semelhante a especificação funcional encontrado no $\mathcal{M}$oise$^+$, e a Rede de Petri é empregada como ferramenta para modelagem e análise dos resultados, mas incluíndo agora o nível social no SMA. O trabalho apresenta-se em uma etapa que os resultados são apenas iniciais, com a evolução da pesquisa podem ainda ocorrer mudanças no método.

O texto deste trabalho está organizado da seguinte forma. A seção 2 apresenta uma revisão sobre temas com importância para o entendimento do trabalho. Na seção 3 é feita uma revisão de trabalhos relacionados à testes e testabilidade em sistemas multiagentes. A seção 4 descreve a proposta do trabalho, e a seção 5 apresenta as conclusões do trabalho.

2. Referencial Teórico

2.1. Sistema Multiagente

De acordo com [Lesser 1999] sistemas multiagentes são sistemas computacionais em que dois ou mais agentes interagem ou trabalham em conjunto para executar um grupo de tarefas, ou para satisfazer um conjunto de metas. O comportamento destes agentes pode ser limitado através de políticas e/ou de uma organização, em que agentes específicos exercem certas funções dentro desta sociedade.

2.2. Organização em SMA

No início, os sistemas tinham apenas uma visão centrada nos aspectos individuais dos agentes, de modo que o SMA era projetado em termos de estados mentais dele, como as crenças, intenções e objetivos, conhecida como metodologia orientada a agentes. A partir de então os SMA evoluíram para uma metodologia orientada à organização, levando em conta suas principais metas, estrutura e normas sociais[Argente et al. 2006]. Os modelos de organização tornaram-se populares para coordenar entidades autônomas em sistemas abertos, descentralizados e dinâmicos. Estes modelos propõem uma regulação dos SMA por um conjunto de normas, planos, mecanismos e/ou estruturas formalmente especificadas para alcançar algum objetivo global desejado.

Existem vários modelos de organização, e eles estruturam o comportamento de entidades complexas em uma hierarquia de entidades encapsuladas, onde cada membro

tem funções a desempenhar [Argente et al. 2006]. A organização pode ajudar grupos de agentes simples a exibir comportamentos complexos e ajudar agentes sofisticados a reduzir a complexidade de seus raciocínios.

2.2.1. $\mathcal{M}$OISE$^+$

O modelo de organização $\mathcal{M}$OISE (*Model of Organization for multI-agent SystEms*) foi inicialmente proposto por [Hannoun et al. 2000] e extensões posteriormente foram desenvolvidas, entre elas [Hübner 2003] e [Hübner et al. 2005]. Este modelo apresenta uma visão centrada na organização, e a organização é como um conjunto normativo de regras que restringe o comportamento dos agentes. Neste consideram-se três formas de representar as restrições organizacionais: papéis, planos e normas [Hannoun et al. 2000].

De acordo com [Hannoun et al. 2000] quando um agente entra em uma organização ele passa a ter que respeitar obrigações e interdições, e tem permissões relacionadas a esta organização. O $\mathcal{M}$OISE é estruturado em três níveis: nível individual, nível coletivo e nível social. No nível individual as restrições são sobre as possibilidades de ação de cada agente. No nível coletivo a restrição está no conjunto de agentes que podem cooperar. No nível social, os enlaces organizacionais restringem os tipos de interação que os agentes podem ter com o sistema.

A Especificação Organizacional (OS) no $\mathcal{M}$oise$^+$ é formada com base nas especificações estrutural, funcional e deôntica [Hübner 2003]:

- *Especificação Estrutural (EE)*: no nível individual, os papéis têm a função de ser elo entre o agente e a organização. No nível social os papéis se relacionam com outros papéis através de ligações e compatibilidade. No nível coletivo os grupos representam um conjunto de agentes com maior afinidade e objetivos mais próximos. A Figura 1 representa o grupo *seleção* os *papéis* (docente, membro, funcionário secretário, presidente e candidato) e a *ligação* entre eles, estre grupo forma uma sociedade (soc) de uma comissão de seleção para o ingresso em um curso de pós-graduação.

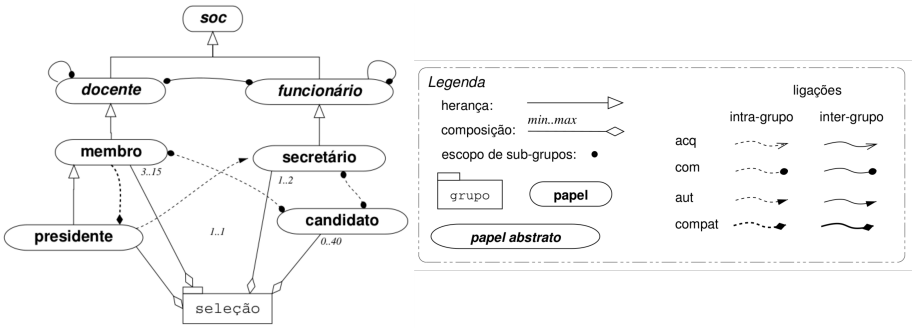

Figura 1. Especificação Estrutural [Hübner 2003].

- *Especificação Funcional (EF)*: é constituída por um conjunto de esquemas sociais e de uma relação entre preferência entre missões. Nos esquemas sociais a meta global é um conceito fundamental. No nível individual um esquema social é constituído por missões. Uma missão é o conjunto de metas globais que podem ser

passadas a um agente através de um de seus papéis. No nível coletivo um esquema social é uma árvore de decomposição de metas globais, a raiz é meta do esquema social e a decomposição de metas é feita através de planos. A Figura 2 representa o esquema social para o exemplo. A leitura do esquema social deve ser feita da esquerda para a direita, o operador *sequência* significa que a meta $g2$ só pode ser realizada quando a meta $g1$ for satisfeita. O operador *escolha* significa que a meta $g7$ será satisfeita se uma e somente uma das metas $g8$ ou $g9$ for satisfeita. O operador *paralelismo*, significa que a meta $g4$ será satisfeita quando ambas as metas $g5$ e $g6$ forem alcançadas, mas as duas sub-metas podem ser buscadas em paralelo.

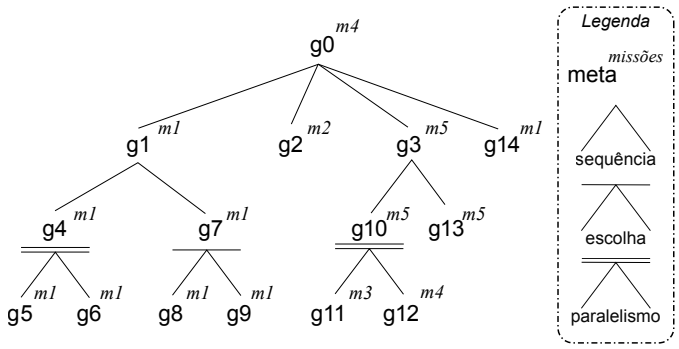

Figura 2. Esquema Social, adaptado de [Hübner 2003].

Os diferentes papéis do exemplo possuem diferentes missões, um agente no papel de candidato, tem a missão $m1$. Para cumprir esta missão o candidato deve ter como objetivo as metas ($g1, g4, g5, g6, g7, g8, g9$ e $g14$), a descrição das metas está na Tabela 1.

Tabela 1. Descrição das metas [Hübner 2003].

meta	descrição	meta	descrição
g0	candidato é aceito no programa de pós-graduação	g7	a inscrição está submetida
g1(Dt)	a documentação é recebida no prazo	g8	submissão eletrônica
g2	a documentação está correta	g9	submissão por correio
g3	candidato é aprovado pela comissão	g10	metas g11 e g12 são cumpridas
g4	metas g3 e g4 são cumpridas	g11	uma reunião está marcada
g5	candidato tem toda a documentação necessária	g12	um relator está indicado
g6	candidato tem um orientador	g13	o projeto do candidato é avaliado
g14	formulário de matrícula preenchido é recebido		

- *Especificação Deôntica (ED)*: é a especificação que relaciona a EE com a EF no nível individual, especificando quais as missões um papel tem permissão ou obrigação de realizar.

2.3. Teste de Software

O teste faz parte de um método de verificação e validação de software onde a verificação investiga se o software atende aos seus requisitos funcionais e não funcionais enquanto a validação é um processo mais amplo no qual o objetivo é garantir que o software atenda às

expectativas do cliente [Sommerville 2010]. O teste tem a função de medir a qualidade do software por pelo menos três termos: do número de defeitos encontrados, pelo rigor dos testes executados e pela cobertura do teste em relação ao sistema [Graham et al. 2008].

Segundo [Myers et al. 2011] é de grande importância ter um bom planejamento dos casos de teste, já que é impossível realizar um teste completo. Uma boa estratégia é tentar fazer o teste o mais completo possível dada as restrições de tempo e custo.

Para encontrar um equilíbrio entre o menor número de casos de testes e uma ótima cobertura do programa são adotadas estratégias de teste de software. Uma estratégia para testes de software fornece um roteiro que descreve as etapas a serem realizadas como parte do teste, quando as etapas são planejadas, realizadas, e quanto esforço, tempo e recursos serão necessários [Pressman 2005].

Diferentes abordagens podem ser utilizadas para identificar casos de testes, entre elas os testes estruturais, são baseados em uma análise dos detalhes procedurais, os caminhos lógicos através do software e as colaborações entre componentes [Jorgensen 2016, Pressman 2005].

2.4. Teste de Software em SMA

Os agentes e sistemas multiagentes possuem uma série de especificidades que tornam o processo de teste mais complexo e que devem abordar algumas questões que não eram preocupação no desenvolvimento de software orientado a objetos. [Rouff 2002, Houhamdi 2011, Nguyen 2009] citam as propriedade do agente ou SMA como ser distribuído e assíncrono, autônomos, trabalhar com envio de mensagens, possuir fatores ambientais e normativos.

Testar uma comunidade de agentes torna os objetivos de testes mais amplos, tendo que verificar se os agentes da comunidade trabalham juntos como projetado, testando a troca de comunicação entre eles, verificando se essa troca de mensagem realmente está ocorrendo entre os agentes previstos e com o ambiente, mas agora com um agravante de ter um número muito maior de interações.

2.5. Rede de Petri

Redes Petri (RP) é uma ferramenta gráfica e matemática para a descrição e análise de processos concorrentes, assíncronos e paralelos que surgem em sistemas distribuídos. Como ferramenta gráfica ela pode auxiliar na comunicação visual como um fluxograma e sendo uma ferramenta matemática, é possível estabelecer equações de estados, equações algébricas e outros modelos matemáticos que regem o comportamento dos sistemas [Murata 1989].

A Figura 3 representa uma RP simples. O grafo da rede modela as propriedades estáticas de um sistema, assim como um fluxograma representa as propriedades estáticas de um programa de computador. O grafo contém dois tipos de nós: os círculos chamados de *lugares* e as barras, chamadas de *transições*. Os nós são conectados por arcos direcionados de *lugares* para *transições* e de *transições* para *lugares* [Peterson 1977].

Além destes dois elementos, uma RP ainda possui propriedades dinâmicas que são resultantes da sua execução. O elemento que atribui esta propriedade dinâmica é a *ficha* [Peterson 1977]. As fichas são indicadas por um ponto em um lugar. A *ficha* pode

representar um recurso em uma posição, ou uma estrutura de dados que se manipula por exemplo [Cardoso and Valette 1997].

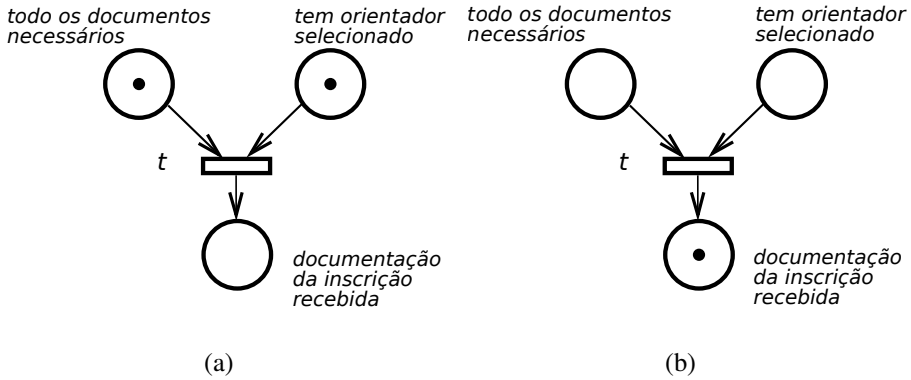

Figura 3. Rede de Petri

A dinâmica da RP atua da seguinte maneira. As fichas são movidas pelo disparo das transições associadas da rede. Na Figura 3(a) podemos observar que temos uma ficha no lugar *todos os documentos necessários* e outra ficha em *tem orientador selecionado*. A ocorrência do evento, associado à transição *t* só pode ocorrer se houver ao menos uma ficha em cada um destes lugares. O disparo da transição *t* retira uma das fichas vinculadas a cada um dos lugares de entrada e as posiciona no lugar de saída *documentação da inscrição recebida* Figura 3(b). A Figura 3 representa estados diferentes do mesmo sistema, uma evolução da rede.

3. Trabalhos Relacionados

Diversos trabalhos utilizam RP no domínio de SMA. Em [Köhler et al. 2001] Redes de Petri Coloridas (RPC) executáveis foram utilizadas para modelar a estrutura e o comportamento dos agentes. [Weyns and Holvoet 2002] relataram como principais motivos da utilização de RPC como ferramenta de modelagem, a visão conceitual clara sobre os agentes e o ambiente e o ótimo suporte a verificação e formalização. [Bai et al. 2004] apresentaram uma abordagem baseada em RPC para formar protocolos de interação flexíveis entre agentes. No trabalho [de Almeida et al. 2004] é introduzido um modelo formal para verificar os planos em um SMA, baseado na modelagem, simulação e verificação do modelo de RP Coloridas Hierárquicas (RPCH). [Poutakidis et al. 2009] apresentam ferramentas para geração de casos de teste para teste de unidade e outra para depuração e monitoramento de sistemas de agentes em execução. [Goncalves 2010] expõem um modelo de RP desenvolvida para especificar o conhecimento em agentes e SMA, independentemente de estruturas e formalismos de representação do conhecimento. [Miller et al. 2011] especifica, e mostra como medir, o grau de detalhes de um conjunto de casos de teste através de um agente de depuração que age como um oráculo, para avaliar a correção de um teste e utilizar a representação da RP do agente como suporte para medidas de cobertura de teste.

Em [Athamena and Houhamdi 2012] os autores utilizam o *Multiagent Systems Engineering* (MaSE), uma metodologia de engenharia de software orientada a agente, que é uma extensão da abordagem orientada a objetos, e propõem uma abordagem baseada em RP para o teste de comportamento de SMA. Para isso os autores desenvolveram

um modelo para a transformação dos diagrama de classe de agente e diagrama de função do MaSE, para o diagrama de sequência do modelo UML 2.0, e então propuseram um modelo para transformar os diagramas de sequencia em RP equivalentes e enfim podem ser aplicados as técnicas de testes automáticos no SMA. Segundo os autores as principais contribuições do trabalho foi propor um processo de teste completo e abrangente para o SMA e reduzir os efeitos colaterais na execução e monitoramento do teste, não utilizando agentes testadores ou agentes oráculos, o que pode influenciar no comportamento dos agentes ou desempenho do sistema.

Em [Winikoff and Cranefield 2014] o objetivo é avaliar o quão difícil é testar um programa de agente na arquitetura BDI. Para isso o autor utiliza a técnica de testes de caixa-branca, baseado no fluxo de controle, um critério básico e muito longo para avaliar a adequação de um conjunto de testes em que todos os caminhos (*All-Path*) do programa sejam cobertos. Um motivo para a escolha de todos os caminhos é que os SMA geralmente envolvem ambientes não-episódicos, sendo que o comportamento de um determinado plano ou meta é geralmente sensível ao histórico do agente, precisando assim considerar as diferentes histórias possíveis.

Os eventos e planos podem ser visualizados como uma árvore em que cada objetivo tem como filhos as instâncias do plano que são aplicáveis a ele, e cada instância do plano tem como filhos os sub-objetivos que ele publica. Esta forma de visualização facilita a análise do número de caminhos através de um programa BDI. Com isso, o programa é visualizado como uma transformação de dados de uma árvore de planos-metas (finita) em uma sequencia de execuções de ações. Assim, a questão de quão grande é o espaço de comportamento para os agentes BDI é respondida derivando fórmulas que permitem calcular o número de comportamentos, bem-sucedidos e mal sucedidos (ou seja, falhou) para uma determinada árvore de planos-metas.

Neste artigo os autores concluíram que um teste completo em um sistema BDI não é viável. O tamanho de espaços de comportamento é muito grande e ainda se torna significativamente maior quando o sistema tem suporte à tolerância de falhas. Os autores também chegaram à conclusão de que o mecanismo de recuperação de falhas é eficaz para alcançar uma baixa taxa de falha real. E novas abordagens foram propostas para lidar com a testabilidade do sistema.

O trabalho [Winikoff 2017] retorna com o objetivo de avaliar se é possível obter garantias em um sistema multiagente através de testes, verificando a testabilidade de um programa, dando continuidade no trabalho anterior [Winikoff and Cranefield 2014], mas utilizando novas métricas. Este trabalho tem por objetivo quantificar quantos testes são necessários para testar um programa de agente BDI para satisfazer um critério considerando o critério de adequação do teste de todas as arestas, que é considerado como o mínimo geralmente aceito [Jorgensen 2016].

Uma das grandes contribuições dos autores é que a análise é genérica permitindo a aplicação a todos os programas que utilizam a arquitetura BDI. Para esta análise equações são derivadas para chegar a conclusão de quantos casos de teste (caminhos) são necessários para cobrir todas as arestas no grafo de fluxo de controle correspondente a um determinado programa BDI.

A Figura 4 apresenta um exemplo de um grafo de controle de fluxo de um pro-

grama BDI, este programa tem início em "S" e possui quatro ações $\alpha_1, \alpha_2, \alpha_3$ e α_4. Caso alguma das ações forem bem-sucedidas então o programa executa "Y" e é encerrado em "E" concluindo-se com sucesso. Se a ação α_1 falhar ela avança para ação α_2 que pode ter sucesso ou a falha pode ocorrer novamente e assim a próxima ação seria executada. Este programa exigiria 5 testes para cobrir todas as bordas, um teste é onde as quatro ações falham ($S \rightarrow \alpha_1 \rightarrow \alpha_2 \rightarrow \alpha_3 \rightarrow \alpha_4 \rightarrow N \rightarrow E$) e os outros 4 testes é para quando uma ação é bem-sucedida e a anterior falhou.

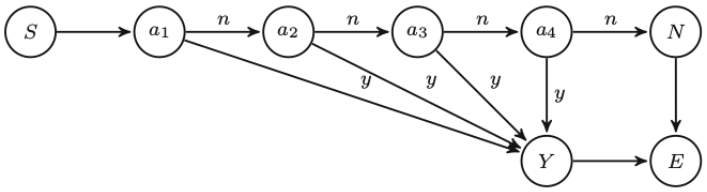

Figura 4. Controle de Fluxo [Winikoff 2017].

Para derivar equações que calculem o menor número de caminhos exigidos de um programa começando em S para chegar em E é necessário descobrir quantos destes caminhos são bem-sucedidos (passando por Y) e quantos falharam (passando por N). Os autores definiram $p(P)$ como o número de caminhos necessários para cobrir todas as arestas do grafo de fluxo de controle correspondente ao programa P, $y(P)$ para os caminhos que vão por Y e $n(P)$ para os caminhos que vão por N, então $p(P) = y(P) + n(P)$.

Logo após os autores consideram $P_1; P_2$, onde um subprograma P_1 é colocado em sequencia com P_2 Figura 5. O subprograma P_1 requer $p(P_1)$ testes para cobrir todas as arestas com $n(P_1)$ testes levando até a falha e $y(P_1)$ levando à uma execução com sucesso.

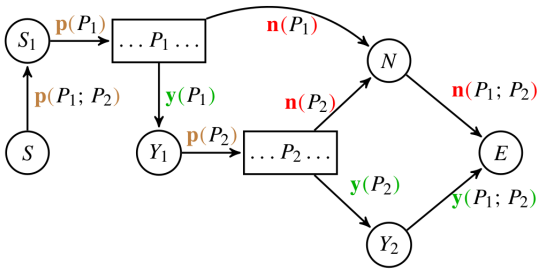

Figura 5. Controle de Fluxo de $P_1; P_2$ [Winikoff 2017].

A partir do exemplo da Figura 5 diversas equações são derivadas para os diferentes casos. Estas equações servem para determinar quantos testes são necessários para garantir uma cobertura adequada em relação ao critério todas as arestas. Então os autores implementam as equação em um programa Prolog que calcula os valores de p(p), y(P) e n(P) para qualquer programa BDI. Para um programa BDI com 62 metas, 2 instâncias de planos aplicáveis e 2 submetas, os resultados encontrado para o [Winikoff and Cranefield 2014] foi 6.33×10^{12} testes que tiveram sucesso e 1.82×10^{13} testes que falharam, enquanto no trabalho [Winikoff 2017] o resultado encontrado é de 141 testes onde se considera todos os planos relevantes de uma meta.

Este trabalho chegou à conclusão de que o número de teste necessário para *Todas Arestas* é muito menor que para a abordagem de *Todos Caminhos*, encontrando resultados onde é possível realizar os testes na prática. Outra conclusão é que permitir o tratamento de exceções não fez diferença significativa no número de testes.

4. Proposta

Em [Winikoff and Cranefield 2014] um grafo de controle de fluxo foi utilizado para avaliar a testabilidade de um sistema BDI. A proposta deste trabalho é avaliar a testabilidade de um SMA utilizando o $\mathcal{M}$oise$^+$ como modelo de organização e empregando rede de Petri como ferramenta de descrição e análise. Partindo como base o trabalho de [Winikoff 2017] o grafo de controle de fluxo apresentado na Figura 4 foi transformado em uma RP Figura 6.

Como indicado pela *ficha* a rede começa no lugar **S**, a transição **t0** é disparada e então a ficha é movida para o lugar **A1**. Neste lugar tem-se uma escolha entre diferentes sequências, onde e a transição **t6** é disparada quando é possível realizar a ação, indo assim para um lugar auxiliar **aux_A1**, sendo assim possível disparar a transição **t10** que leva até ao lugar **Y** que corresponde à execução bem sucedida do programa, e finalmente para **E** que é o encerramento do programa. A transição **t1** é acionada quando o agente não consegue executar a ação, direcionando assim para o *A2*, esta ação pode ser bem sucedida, indo para a transição **t7**, ou falhar e ir para a transição **t2**, operando semelhante ao apresentado no *lugar* **A1**.

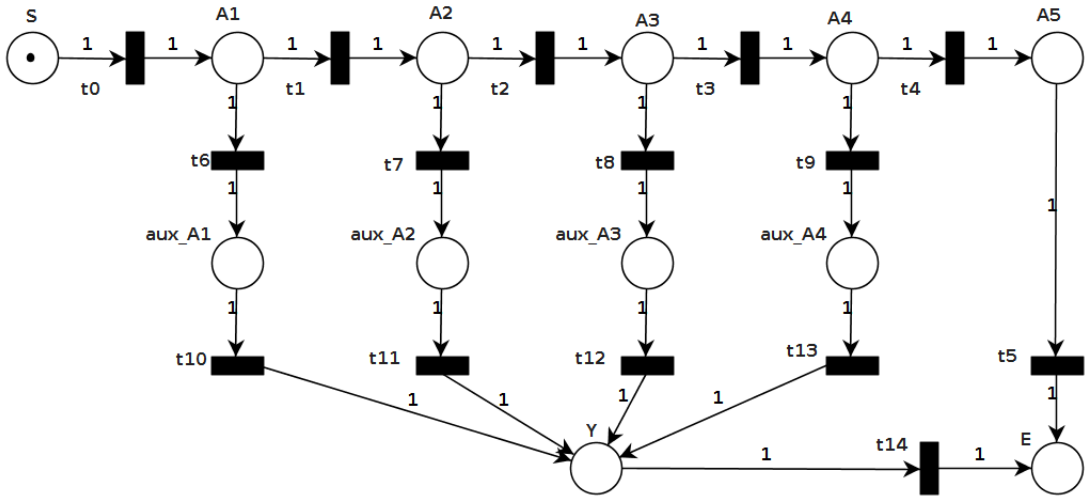

Figura 6. RP do grafo de controle de fluxo.

No método de [Winikoff and Cranefield 2014, Winikoff 2017] a avaliação de testabilidade é realizada apenas no nível de metas e ações. Para apresentar a proposta utilizando o $\mathcal{M}$oise$^+$ será utilizado um exemplo exposto em [Hübner 2003]. Neste contexto temos que avaliar em múltiplos níveis que não estavam previsto em outros trabalhos, como missões, planos, metas e ações.

Primeiro é feita uma relação dos operadores apresentados na legenda da Figura 2 e sua RP equivalente, Figura 7. O operador de *sequência* representa uma cadeia de metas em sequência. A meta **G10** inicia a sequência, caso a transição **t1** for iniciada a

ficha é retirada de G10 e é colocada em **G13**, permitindo assim o seguimento da cadeia. No operador *escolha* apenas uma das metas, **G8** ou **G9** podem ser adotadas, para isso o lugar **G7** recebe uma restrição, quando uma transição for disparada e a ficha for inserida no lugar **G7**, a outra transição será desabilitada, não permitindo assim que a outra meta seja realizada. O operador *paralelismo* significa que as metas **G5** e **G6** podem progredir paralelamente e de modo assíncrono, mas a transição **t1** só será disparada quando as duas metas estiverem concluídas.

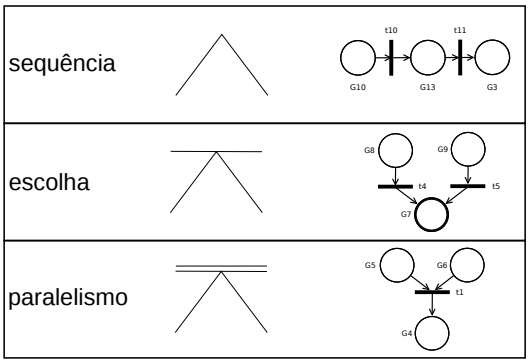

Figura 7. Relações dos operadores para Rede de Petri.

Iniciando pela árvore de metas globais da Figura 2, e utilizando os operadores da Figura 7 a árvore é transformada em RP para cada missão, ainda de maneira isolada Figura 8. O processo começa lendo a árvore da esquerda para a direita da parte inferior, e transformando as metas em lugares da RP, utilizando os operadores relacionados, até que um conjuntos de meta para finalizar a missão seja concluída.

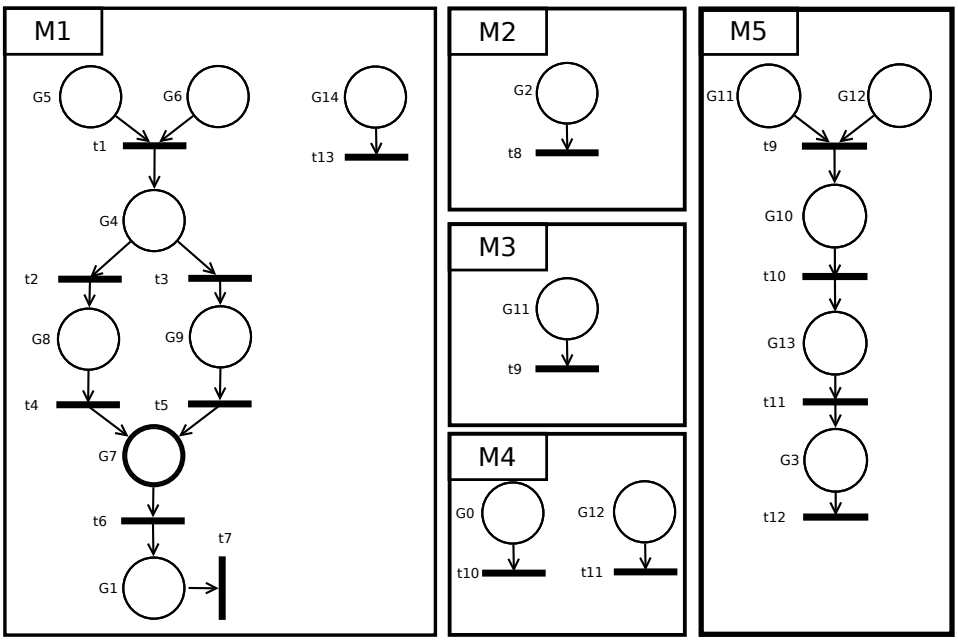

Figura 8. Transformação das missões do exemplo em RP.

5. Conclusões

Este trabalho apresenta uma proposta para avaliar a testabilidade de um SMA, que utiliza o modelo de organização $\mathcal{M}$oise$^+$ utilizando RP. O trabalho foi baseado inicialmente na abordagem proposta por [Winikoff 2017] onde as GPT representam a execução de um programa BDI e os caminhos (todos ou as arestas) representam o número de testes a serem realizados para a cobertura do sistema.

Até o momento é apresentada uma proposta para a transformação do esquema sociais, representado pela decomposição de metas globais em uma RP, formando assim a descrição das missões de maneira isolada. A próxima etapa é descrever um nível de descrição que estabelece as relações entre as missões, onde juntando as RP que representam $m1$, $m2$, $m3$, $m4$ e $m5$ o resultado é $g0$. Para cada meta os agentes ainda possuem planos e estes planos possuem ações. Esta relação de subníveis também será descrita e apresentada em redes de Petri.

Referências

Argente, E., Julian, V., and Botti, V. (2006). Multi-agent system development based on organizations. *Electronic Notes in Theoretical Computer Science*, 150(3):55–71.

Athamena, B. and Houhamdi, Z. (2012). A petri net based multi-agent system behavioral testing. *Modern Applied Science*, 6(3):46.

Bai, Q., Zhang, M., and Win, K. T. (2004). A colored petri net based approach for multi-agent interactions. In *Proc. of 2nd International Conference on Autonomous Robots and Agents, Palmerston North, New Zealand*, pages 152–157.

Cardoso, J. and Valette, R. (1997). *Redes de petri*. Editora da UFSC.

de Almeida, H. O., da Silva, L. D., Perkusich, A., and de Barros Costa, E. (2004). A formal approach for the modelling and verification of multiagent plans based on model checking and petri nets. In *International Workshop on Software Engineering for Large-Scale Multi-agent Systems*, pages 162–179. Springer.

Goncalves, E. M. N. (2010). An approach to specify knowledge in multi-agent systems using petri nets. In *Network and System Security (NSS), 2010 4th International Conference on*, pages 456–461. IEEE.

Graham, D., Van Veenendaal, E., and Evans, I. (2008). *Foundations of software testing: ISTQB certification*. Cengage Learning EMEA.

Hannoun, M., Boissier, O., Sichman, J., and Sayettat, C. (2000). Moise: An organizational model for multi-agent systems. *Advances in Artificial Intelligence*, pages 156–165.

Houhamdi, Z. (2011). Multi-agent system testing: A survey. *International Journal of Advanced Computer Science and Applications*, 2(6):135–141.

Hübner, J. F. (2003). *Um modelo de reorganização de sistemas multiagentes.* PhD thesis, Universidade de São Paulo.

Hübner, J. F., Sichman, J. S., and Boissier, O. (2005). \mathcal {SM} oiseˆ{+}: A middleware for developing organised multi-agent systems. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 64–77. Springer.

Hübner, J. F., Sichman, J. S., and Boissier, O. (2007). Developing organised multiagent systems using the moise+ model: programming issues at the system and agent levels. *International Journal of Agent-Oriented Software Engineering*, 1(3-4):370–395.

Jorgensen, P. C. (2016). *Software testing: a craftsman's approach*. CRC press.

Köhler, M., Moldt, D., and Rölke, H. (2001). Modelling the structure and behaviour of petri net agents. *Applications and Theory of Petri Nets 2001*, pages 224–241.

Lesser, V. R. (1999). Cooperative multiagent systems: A personal view of the state of the art. *IEEE Transactions on knowledge and data engineering*, 11(1):133–142.

Miller, T., Padgham, L., and Thangarajah, J. (2011). Test coverage criteria for agent interaction testing. In *Agent-oriented Software Engineering XI*, pages 91–105. Springer.

Murata, T. (1989). Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580.

Myers, G. J., Sandler, C., and Badgett, T. (2011). *The art of software testing*. John Wiley & Sons.

Nguyen, D. C. (2009). *Testing techniques for software agents*. PhD thesis, University of Trento.

Peterson, J. L. (1977). Petri nets. *ACM Computing Surveys (CSUR)*, 9(3):223–252.

Poutakidis, D., Winikoff, M., Padgham, L., and Zhang, Z. (2009). Debugging and testing of multi-agent systems using design artefacts. *Multi-Agent Programming:*, pages 215–258.

Pressman, R. S. (2005). *Software engineering: a practitioner's approach*. Palgrave Macmillan.

Rouff, C. (2002). A test agent for testing agents and their communities. In *Aerospace Conference Proceedings, 2002. IEEE*, volume 5, pages 5–2638. IEEE.

Sommerville, I. (2010). *Software Engineering*. Addison-Wesley Publishing Company, USA, 9th edition.

Van Den Broek, E. L., Jonker, C. M., Sharpanskykh, A., Treur, J., et al. (2005). Formal modeling and analysis of organizations. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 18–34. Springer.

Weyns, D. and Holvoet, T. (2002). A colored petri-net for a multi-agent application. In *Proceedings of MOCA'02*, volume 561, pages 121–141.

Winikoff, M. (2010). Assurance of agent systems: What role should formal verification play? *Specification and Verification of Multi-agent systems*, pages 353–383.

Winikoff, M. (2017). Bdi agent testability revisited. *Autonomous Agents and Multi-Agent Systems*, pages 1–39.

Winikoff, M. and Cranefield, S. (2014). On the testability of bdi agent systems. *Journal of Artificial Intelligence Research*, 51:71–131.