Uma Revisão sobre Testes em Sistemas Multiagentes

Ricardo A. Machado
Centro de Ciências Computacionais
Universidade Federal do Rio Grande (FURG)
Rio Grande – RS – Brazil
Email: ricardoarend@gmail.com

Eder M. Gonçalves
Centro de Ciências Computacionais
Universidade Federal do Rio Grande (FURG)
Rio Grande – RS – Brazil
Email: edergoncalves@furg.br

***Resumo*—Como se sabe todo sistema de software precisa ser devidamente testado antes de entrar no mercado para que haja uma garantia de seu funcionamento e uma segurança ao usuário. 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 teste necessárias para sua completa validação.**

Esse artigo apresenta uma revisão e classificação da literaturas que tenham como foco os testes em SMA. As motivações que levaram a escolha do conteúdo aqui pesquisado foram baseadas na busca de respostas para as seguintes questões: quais as dificuldades existentes para testar um SMA, que níveis de teste podem ser realizados e quais técnicas são mais utilizadas para gerar casos de teste. Os objetivos são, além de responder tais questões, analisar e descrever as principais abordagens existentes para geração e execução de testes nesses sistemas.

I. Introdução

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. Um sistema multiagente (SMA) é uma sociedade de agentes autônomos, que evolui em cooperação em seu ambiente para atingir coletivamente objetivos globais [1].

Uma das grandes dificuldades ao trabalhar com SMA está relacionado com a realização de testes. O teste é uma atividade de desenvolvimento de software dedicada a avaliar a qualidade do produto e melhorá-lo, identificando defeitos e problemas [2]. Em sistemas multiagentes, o teste é uma tarefa desafiadora, e requer novas técnicas que lidem com sua natureza específica. Segundo Houhamdi [3] as principais razões para essas dificuldades são:

- maior complexidade, pois existem vários processos distribuídos que são executados de forma autônoma e simultânea;
- quantidade de dados, pois os sistemas podem ser compostos por milhares de agentes, cada um com seus próprios dados;
- efeito de irreprodutibilidade, já que não há garantia que duas execuções do mesmo sistema com as mesmas entradas levem ao mesmo estado, dificultando a procura de erros;
- eles são não-determinísticos, uma vez que não é possível determinar antecipadamente todas as interações de um agente durante sua execução.

Com o objetivo de responder algumas perguntas de pesquisa e gerar uma fundamentação para futuros trabalhos foi feita uma revisão de literatura tendo como base o teste de SMA. Para tal foi adotada uma metodologia que é descrita na seção II com os passos para a geração da busca e seleção dos artigos. Na seção III os artigos selecionados são classificados e descritos. Por fim, a seção IV traz uma conclusão do trabalho realizado a partir das respostas encontradas para as perguntas de pesquisa.

II. Metodologia da Revisão

A. Perguntas de Pesquisa

O principal objetivo desta revisão sistemática da literatura é reconhecer e categorizar a literatura existente sobre abordagens de geração de casos de teste em SMA. Portanto, as questões de pesquisa abordadas nesta revisão são:

1) Quais as dificuldades para testar um sistema multiagente?
2) Que níveis de teste são os mais abordados na literatura?
3) Quais são as principais técnicas para geração de casos de teste em sistemas multiagentes?

B. Protocolo da Busca

Uma vez com as perguntas definidas, a próxima etapa foi gerar uma *string* para iniciar as buscas. O processo para realizar as buscas da pesquisa teve os seguintes passos:

1) Definir os termos primários da busca que são: "*agent system*", "*multi-agent system*", "*multiagent system*".
2) Identificar termos específicos que tenham relação direta com o objetivo do trabalho sendo: "*test case*".
3) Identificar os termos usados para descrever os resultados da pesquisa: "*correction*", "*verification*", "*validation*", "*error*", "*faults*", "*failure*".
4) Utilizar os Booleanos *OR* e *AND* para concatenar as palavras acima listadas de forma que gerasse uma busca satisfatória.

Os termos de pesquisa resultantes são descritos através de uma *string* que ficou da seguinte maneira: *("agent system"OR*

"multi-agent system"OR "multiagent system") AND ("test case") AND (correction OR verification OR validation OR error OR faults OR failure).

Com a *string* definida foram realizadas as buscas nos repositórios do Google Scholar, Springer e IEEE sendo que essa *string* foi utilizada exatamente da maneira acima descrita. O motivo da escolha do Scholar foi a grande diversidade de outros repositórios indexada nele. No caso do Springer foi encontrados diferentes artigos que não haviam sido anteriormente retornados pela busca do Scholar. O IEEE foi escolhido para aumentar abrangência da busca. O período analisado foi entre o ano de 2007 até setembro de 2019, data da publicação deste artigo.

C. Seleção dos Artigos

Na busca foram encontrados 2725 resultados sendo 2190 no Scholar, 730 no Springer e apenas 5 no IEEE. Para a seleção dos artigos mais relevantes foram adotados critérios de inclusão e exclusão sendo eles listados a seguir:

Inclusão:

1) Artigos que satisfazem as palavras chaves de busca e tratam das questões de pesquisa.
2) Artigos que tratam de testes em sistemas multiagentes ou agentes.
3) Artigos que apresentam uma técnica ou ferramenta para geração de casos de teste em sistemas multiagentes.

Exclusão:

1) Artigos que apresentam sistemas multiagentes mas não possuem nenhum objetivo ou método com finalidade de testar esses sistemas.
2) Artigos que tratam de teste de softwares mas não com foco em sistemas multiagentes ou em agentes.

Os critérios acima foram utilizados como filtros através da leitura do título e do *abstract* de cada artigo. Por exemplo determinados artigos encontrados apresentavam alguma metodologia para testes de software porém voltados para sistemas convencionais e não SMA, fazendo com que fossem descartados da busca. No final dessa primeira triagem restaram um total de 36 artigos. Em seguida foram descartados artigos de teses assim como artigos duplicados em ambos repositórios ou mesmo sequências anteriores de um mesmo trabalho. Por fim restaram 20 artigos sendo 15 do Scholar e 5 do Springer que serão apresentados detalhadamente no capítulo seguinte.

III. Classificação dos Resultados

A. Níveis de Classificação

Como existem diferentes níveis de testes que podem ser aplicados num sistema de agentes se torna necessário classificar os resultados aqui obtidos de maneira organizada com base nesses níveis. Uma classificação anterior foi descrita por Nguyen [4] 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.

Sendo a classificação acima descrita bem detalhada foi entendido que ela é válida para essa revisão. Portanto os artigos foram organizados com base nessa classificação, não havendo necessidade de criar algo novo para esse trabalho.

B. Classificação dos Artigos

A Tabela I apresenta uma classificação das publicações de acordo com os níveis descritos na seção anterior. 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.

Tabela I
Classificação dos Artigos

Unidade	[5], [6], [7], [8], [9].
Agente	[1], [2], [9], [10], [11], [12], [13], [14], [15], [16], [17].
Integração	[1], [2], [5], [6], [8], [12], [15], [16], [17], [18], [19], [20], [21].
Sistema	[2], [5], [8], [9], [22].
Aceitação	[1]

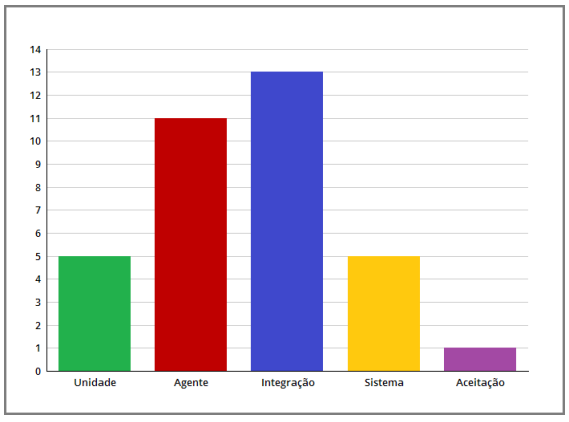

Figura 1. Níveis de Teste

A Figura 1 mostra um gráfico com a quantidade de publicações para cada nível de teste. O objetivo desse gráfico é

facilitar a visualização da informação gerada por essa revisão. Os testes de unidade e de sistema foram utilizados em cinco artigos cada um, os testes de agente foram onze, integração treze, e o teste de aceitação foi identificado em apenas um artigo. Nota-se que existe maior foco nos testes de agente e de integração. Como já foi comentado um mesmo artigo pode fazer uso de vários níveis o que fez com que o total de artigos selecionados não coincida com o total listado no gráfico.

Na Figura 2 podemos visualizar a quantidade de artigos selecionados por ano de publicação, com o objetivo de avaliar a quantidade de contribuições geradas ao longo do tempo. Como é possível notar os três primeiros anos pesquisados tiveram maior número de resultados encontrados com oito artigos. Já entre 2010 a 2016 os valores se mantiveram baixos, tendo sido selecionados apenas sete artigos nesse período. No ano de 2017 houve um aumento para três , já em 2018 nenhum artigo encontrado e em 2019 novamente encontrados três artigos.

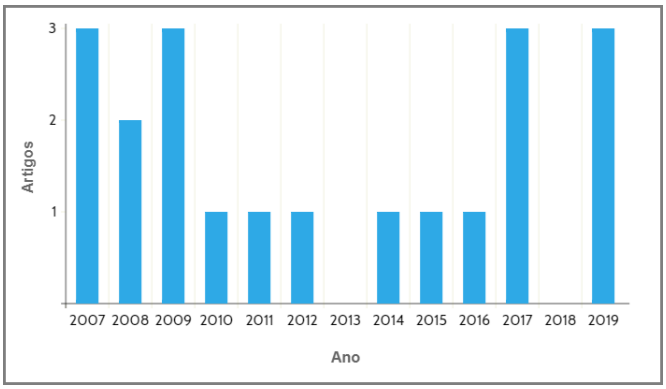

Figura 2. Linha do tempo

C. Resumo dos Artigos

A seguir serão apresentados os artigos selecionados nessa revisão com uma breve descrição dos mesmos.

Kissoum and Sahnoun [10] descreve uma abordagem formal para especificar, desenvolver e testar SMA. Esta abordagem baseia-se na utilização da linguagem algébrica de especificação formal Maude [23], 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 à interação do agente e seu comportamento interno.

Coelho et al. [5] apresenta 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. [11] desenvolve 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 et al. [13] define 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. [12] apresenta avanços em testes e depuração feitos pela metodologia INGENIAS [24]. Baseado numa abordagem direcionada por modelos ele 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 [8] 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. [6] apresenta um *framework* para teste e depuração de SMA baseado no uso de artefatos de design. Mais precisamente foi demonstrado 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 et al. [7] descreve 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 et al. [18] define 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 and Athamena [22] apresenta uma abordagem para testes estruturais em SMA especificando um processo de teste que complementa a metodologia orientada a metas chamada Tropos [25]. 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 and Zhu [14] propôs um *framework* de automação de testes chamado CATest utilizando a linguagem de especificação formal baseada em agentes SLABS [26]. 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 inexistência 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. [15] desenvolveu 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.

Ur Rehman and Nadeem [19] apresenta uma abordagem para teste em SMA baseado em design de artefatos da ferramenta Prometheus [27]. 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. [2] propôs 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.

Do Nascimento et al. [16] modelou e desenvolveu uma arquitetura baseada em publicação de assinaturas para facilitar a implementação de sistemas que testem os SMA nos níveis de agente e grupo. O autor utilizou uma plataforma chamada RabbitMQ [28] 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 [9], com foco em programas de agentes BDI, o autor 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 necessário entre os critérios de 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. [1] apresenta 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 o autor define quais as metas que devem ser alcançadas para que o teste seja bem sucedido. Como 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 et al. [20] apresenta 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 [9] 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.

Benac Earle and Fredlund [17] descreve um *framework* para testar SMA escritos na linguagem de programação Jason, usando a técnica de teste conhecida como teste baseado em propriedades. Essa técnica consiste em gerar automaticamente casos de teste, a partir de uma descrição mais abstrata do comportamento do sistema em teste: as propriedades. A abordagem substitui um subconjunto dos agentes no sistema multiagente por uma máquina de estados *QuickCheck* [29]. Por sua vez a máquina de estados *QuickCheck* desempenha o papel

desses agentes, interagindo com os agentes reais restantes, enviando mensagens e modificando o ambiente, e julgando se esses agentes reais restantes são implementados corretamente, examinando as mensagens enviadas para qualquer agente substituído e as percepções de crença que eles recebem.

Dehimi and Mokhati [21] propõe uma abordagem de teste baseado em modelo para testar as interações entre agentes. A abordagem usa um diagrama de sequência AUML como modelo e restrições expressas na linguagem OCL (*Object Constraint Language*). A abordagem gera um conjunto de casos de teste capazes de, individualmente, abranger interações entre agentes, bem como possíveis cenários que podem ser executados de maneira inclusiva, exclusiva ou paralela.

IV. Conclusão

Esse artigo apresentou como revisão uma série de abordagens que foram desenvolvidas nos últimos anos que tivessem como foco principal uma solução, seja ela um modelo, ferramenta ou método, para testar um agente ou um SMA completo. Os resultados da busca foram classificados em diferentes níveis utilizando uma classificação existente na literatura que se mostrou válida e atualizada. Com esses resultados foi possível analisar as evoluções feitas na área de testes durante um período de doze anos.

As perguntas de pesquisa que foram apresentadas no capítulo II serão respondidas a seguir com base no conteúdo dos artigos selecionados:

1) **Quais as dificuldades para testar um sistema multiagente?** Como visto em [3] devido ao seu comportamento não-determinístico os agentes possuem uma complexidade maior do que um software tradicional. Isso significa que suas ações são pouco previsíveis e para gerar e executar todos os testes necessários é preciso prever todos os caminhos de teste possíveis. Além disso segundo [1] testar um SMA implica não só em monitorar o comportamento individual do agente, mas também a interação entre os agentes e o sistema global precisam ser testados. Por isso testar esses sistemas requer novas técnicas de teste que lidem com sua natureza específica.
2) **Que níveis de teste são os mais abordados na literatura?** Como foi apresentado na Figura 1 os testes a nível de integração foram os mais abordados com um total de treze artigos encontrados seguidos pelos testes de agente com onze. Esse resultado mostra que o foco maior está para a integração, seja dos diferentes componentes de um agente ou dos próprios agentes. Nesse último caso são realizados testes da interação dos agentes com outros agentes, com o ambiente ou da comunicação entre esses agentes.
3) **Quais são as principais técnicas para geração de casos de teste em sistemas multiagente?** Praticamente todos os artigos utilizam como técnica a geração automática de casos de teste através de *frameworks*, isso se deve pelo fato que como os agentes tem um comportamento imprevisível o número de casos de teste possíveis num SMA é muito grande para que seja feito manualmente o que consumiria muito tempo.

Apesar de ser essencial realizar testes em um SMA para garantir seu perfeito funcionamento, ainda existem muitas dificuldades que precisam ser superadas. Além disso o número de artigos publicados nos últimos tempos tem se mantido baixo o que faz com que essa seja uma área de conhecimento que ainda pode ser bastante explorada.

Referências

[1] C. Barnier, A. Mercier, J.-P. Jamont *et al.*, "Toward an embedded multi-agent system methodology and positioning on testing," in *2017 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2017, pp. 239–244.

[2] S. Kerraoui, Y. Kissoum, M. Redjimi, and M. Saker, "Matt: Multi agents testing tool based nets within nets," *Journal of Information and Organizational Sciences*, vol. 40, no. 2, pp. 165–184, 2016.

[3] Z. Houhamdi, "Multi-agent system testing: A survey," *International Journal of Advanced Computer*, 2011.

[4] D. C. Nguyen, "Testing techniques for software agents," Ph.D. dissertation, University of Trento, 2009.

[5] R. Coelho, E. Cirilo, U. Kulesza, A. von Staa, A. Rashid, and C. Lucena, "Jat: A test automation framework for multi-agent systems," in *Software Maintenance, 2007. ICSM 2007. IEEE International Conference on*. IEEE, 2007, pp. 425–434.

[6] D. Poutakidis, M. Winikoff, L. Padgham, and Z. Zhang, "Debugging and testing of multi-agent systems using design artefacts," in *Multi-Agent Programming:*. Springer, 2009, pp. 215–258.

[7] Z. Zhang, J. Thangarajah, and L. Padgham, "Automated testing for intelligent agent systems," in *International Workshop on Agent-Oriented Software Engineering*. Springer, 2009, pp. 66–79.

[8] T. Salamon, "A three-layer approach to testing of multi-agent systems," in *Information Systems Development*. Springer, 2009, pp. 393–401.

[9] M. Winikoff, "Bdi agent testability revisited," *Autonomous Agents and Multi-Agent Systems*, vol. 31, no. 5, pp. 1094–1132, 2017.

[10] Y. Kissoum and Z. Sahnoun, "Test cases generation for multi-agent systems using formal specification," *Computer Systems and Applications*, pp. 76–83, 2007.

[11] C. D. Nguyen, A. Perini, P. Tonella, and F. B. Kessler, "Automated continuous testing of multi-agent systems," in *The fifth European workshop on Multi-agent systems*. Citeseer, 2007.

[12] J. J. Gomez-Sanz, J. Botía, E. Serrano, and J. Pavón, "Testing and debugging of mas interactions with ingenias," in *International Workshop on Agent-Oriented Software Engineering*. Springer, 2008, pp. 199–212.

[13] C. D. Nguyen, A. Perini, and P. Tonella, "Ontology-based test generation for multiagent systems," in *Proceedings of the 7th international joint conference on Autonomous*

agents and multiagent systems-Volume 3. International Foundation for Autonomous Agents and Multiagent Systems, 2008, pp. 1315–1320.

[14] S. Wang and H. Zhu, "Catest: a test automation framework for multi-agent systems," in *Computer Software and Applications Conference (COMPSAC), 2012 IEEE 36th Annual*. IEEE, 2012, pp. 148–157.

[15] F. E. Eassa, L. J. Osterweil, M. A. Fadel, S. Sandokji, and A. Ezz, "Dttas: A dynamic testing tool for agent-based systems," *Pensee Journal*, vol. 76, no. 5, 2014.

[16] N. M. Do Nascimento, C. J. M. Viana, A. von Staa, and C. Lucena, "A publish-subscribe based architecture for testing multiagent systems." in *SEKE*, 2017, pp. 521–526.

[17] C. Benac Earle and L.-Å. Fredlund, "A property-based testing framework for multi-agent systems," in *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2019, pp. 1823–1825.

[18] T. Miller, L. Padgham, and J. Thangarajah, "Test coverage criteria for agent interaction testing," in *International Workshop on Agent-Oriented Software Engineering*. Springer, 2010, pp. 91–105.

[19] S. Ur Rehman and A. Nadeem, "An approach to model based testing of multiagent systems," *The Scientific World Journal*, vol. 2015, 2015.

[20] E. M. Gonçalves, B. C. Rodrigues, and R. A. Machado, "Assessment of testability on multiagent systems developed with organizational model $\mathcal{M}$oise," in *EPIA Conference on Artificial Intelligence*. Springer, 2019, pp. 581–592.

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

[22] Z. Houhamdi and B. Athamena, "Structured system test suite generation process for multi-agent system," *International Journal on Computer Science and Engineering*, vol. 3, no. 4, pp. 1681–1688, 2011.

[23] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martı-Oliet, J. Meseguer, and J. F. Quesada, "Maude: Specification and programming in rewriting logic," *Theoretical Computer Science*, vol. 285, no. 2, pp. 187–243, 2002.

[24] J. Pavón, J. J. Gómez-Sanz, and R. Fuentes, "The ingenias methodology and tools," in *Agent-oriented methodologies*. IGI Global, 2005, pp. 236–276.

[25] F. Giunchiglia, J. Mylopoulos, and A. Perini, "The tropos software development methodology: processes, models and diagrams," in *International Workshop on Agent-Oriented Software Engineering*. Springer, 2002, pp. 162–173.

[26] H. Zhu, "Slabs: A formal specification language for agent-based systems," *International Journal of Software Engineering and Knowledge Engineering*, vol. 11, no. 05, pp. 529–558, 2001.

[27] L. Padgham, J. Thangarajah, and M. Winikoff, "Prometheus design tool," in *AAAI 2008*. AAAI Press, 2008.

[28] A. Richardson *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*, vol. 30, p. 33, 2012.

[29] J. Hughes, "Quickcheck testing for fun and profit," in *International Symposium on Practical Aspects of Declarative Languages*. Springer, 2007, pp. 1–32.