INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO CEARÁ PRÓ-REITORIA DE ENSINO
COORDENADORIA DE CIÊNCIA DA COMPUTAÇÃO DO CAMPUS ARACATI BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Antônio Linco Lima Fernandes

ProMon: Uma Proposta de Monitoramento para Testes de Agentes Racionais

ARACATI-CE
2017

Antônio Linco Lima Fernandes

PROMON: UMA PROPOSTA DE MONITORAMENTO PARA TESTES DE AGENTES RACIONAIS

Trabalho de conclusão de curso apresentado à Coordenadoria de Ciência da Computação do Instituto Federal de Educação, Ciência e Tecnologia do Ceará - Campus Aracati como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Área de pesquisa: Inteligência Artificial, Engenharia de Software.

Orientador: Profa. MSc. Francisca Raquel de Vasconcelos Silveira

Aracati-CE
2017

Dados Internacionais de Catalogação na Publicação (CIP)

F363p Fernandes, Antônio Linco Lima.

Promon: Uma proposta de monitoramento para testes de agentes racionais./ Antônio Linco Lima Fernandes. – Aracati: IFCE, 2017.75f.:

Orientadora: Profª. Msc. Francisca Raquel de Vasconcelos Silveira .
Monografia (Graduação em Ciência da computação) – IFCE.

1. Agente Monitor de Testes. 2. Diagnóstico de Falhas.
3. Teste de Agentes Racionais. I. Título.

IFCE/BIBLIOTECA/ARACATI CDD: 006.3

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO CEARÁ COORDENADORIA DE CIÊNCIA DA COMPUTAÇÃO DO CAMPUS ARACATI BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

ANTÔNIO LINCO LIMA FERNANDES

Este Trabalho de Conclusão de Curso foi julgado adequado para a obtenção do Grau de Bacharel em Ciência da Computação, sendo aprovado pela Coordenadoria de Ciência da Computação do Instituto Federal de Educação, Ciência e Tecnologia do Ceará - Campus Aracati e pela banca examinadora:

Profa. MSc. Francisca Raquel de Vasconcelos Silveira
Instituto Federal do Ceará - IFCE
Orientadora

MSc. Yrleyjander Salmito Lopes
CPQI Technology
Financial Technical Consultant

Prof. Esp. Fábio José Gomes de Sousa
Instituto Federal do Ceará - IFCE

Aracati, 03 de maio de 2017

Agradecimentos

A Deus por ter me dado força e coragem durante toda esta longa caminhada.

Ao IFCE, seu corpo docente, direção e administração que oportunizaram a janela que hoje vislumbro um horizonte superior, eivado pela acendrada confiança no mérito e ética aqui presentes.

A minha orientadora Profa. MSc. Raquel Silveira, pelo empenho dedicado à elaboração deste trabalho e por todo incentivo que sempre foi dado.

Ao MSc. Yrleyjander Salmito e ao Prof. Esp. Fábio José pelo paciente trabalho de revisão da redação.

Ao meu pai Luzimar, minha mãe Maria e aos meus irmãos, pelo amor, incentivo e apoio incondicional.

Aos meus amigos, que fizeram parte da minha formação e que vão continuar presentes em minha vida.

A todos que direta ou indiretamente fizeram parte da minha formação, o meu muito obrigado.

Dedico este trabalho aos meus pais e aos meus irmãos por todo apoio que me foi dado.

Resumo

A Inteligência Artificial (IA) é um ramo da ciência da computação que propõe elaborar dispositivos que simulam a capacidade humana de raciocinar, perceber, tomar decisões e resolver problemas diversos. Um dos mecanismos de aplicação da IA são os agentes inteligentes, tipos de programa que escolhem a melhor ação possível a partir de uma percepção (ou sequência de percepções). Agentes inteligentes são vistos como uma tecnologia promissora para atender às necessidades empresariais modernas e oferecem também uma metodologia conceitual eficiente para projetar sistemas complexos. Devido ao grau de complexidade e as propriedades dos agentes, o teste de agentes racionais é necessário e, por vezes, desafiador, requerendo que as técnicas de teste existentes para softwares tradicionais sejam adaptadas no sentido de adequar a natureza específica dos agentes, para poder validar o funcionamento desses sistemas. Neste contexto, este trabalho propõe a concepção de um agente de monitoramento (*ProMon*) e diagnóstico das falhas do agente testado, que seja capaz de colaborar com o teste de programas de agentes racionais, gerando para o projetista informações relevantes sobre o desempenho do agente, além da identificação dos estados ideais e de qual módulo de processamento de informação do agente que está causando a falha, permitindo ao projetista realizar melhorias na estrutura interna dos programas agentes.

Palavras-chave: Agente Monitor de Testes. Diagnóstico de Falhas. Teste de Agentes Racionais.

Sumário

Lista de Figuras

Lista de abreviaturas e siglas

ACC	Agent Communication Channel
ACL	Agent Communication Language
AMS	Agent Management System
API	Application Programming Interface
AUT	Agent Under Test
BDI	Belief-Desire-Intention
DF	Directory Facilitator
DNA	DeoxyriboNucleic Acid
EB	Exabytes
ENIAC	Electronic Numerical Integrator and Computer
FIPA	Foundation For Intelligent, Physical Agents
GA	Algoritmo Genético
GUI	Graphical User Interface
IA	Inteligência Artificial
IBM	International Business Machines
IFCE	Instituto Federal de Educação, Ciência e Tecnologia do Ceará
JADE	Java DEvelopment Framework
JMetal	Metaheuristic Algorithms in Java
NSGA	Nondominated Sorting Genetic Algorithms
NSGA-II	Nondominated Sorting Genetic Algorithms-II
PEAS	Performance, Environment, Actuators, Sensors
RMA	Remote Monitoring Agent
TCC	Trabalho de Conclusão de Curso
TI	Tecnologia da Informação
xml	eXtensible Markup Language
ZB	Zettabyte

1. Introdução

Com o advento tecnológico e a velocidade crescente com que ocorrem as mudanças nessa esfera, auxiliados pelo volume de informação, cada vez maior, a que se pode acessar, tem-se criado a necessidade de mecanismos que possam lidar com a multiplicidade acarretada no meio computacional. Nesse âmbito, a criação de softwares com potencialidade estendida e maior nível de autonomia que os convencionais têm ganhado espaço no cenário da Tecnologia da Informação (TI) (STOBING, 2015).

Motivações no cenário contemporâneo para o desenvolvimento de tecnologias mais robustas não são novidade, a agilidade da era moderna torna as pessoas mais sedentas por respostas rápidas e praticidade no dia a dia, segundo o explicado pela Cisco (2016). A tecnologia baseada em Inteligência Artificial (IA), ganha novos horizontes ao entrar como auxílio para soluções amplamente difundidas na rotina cotidiana de centenas de milhares de pessoas (STOBING, 2015).

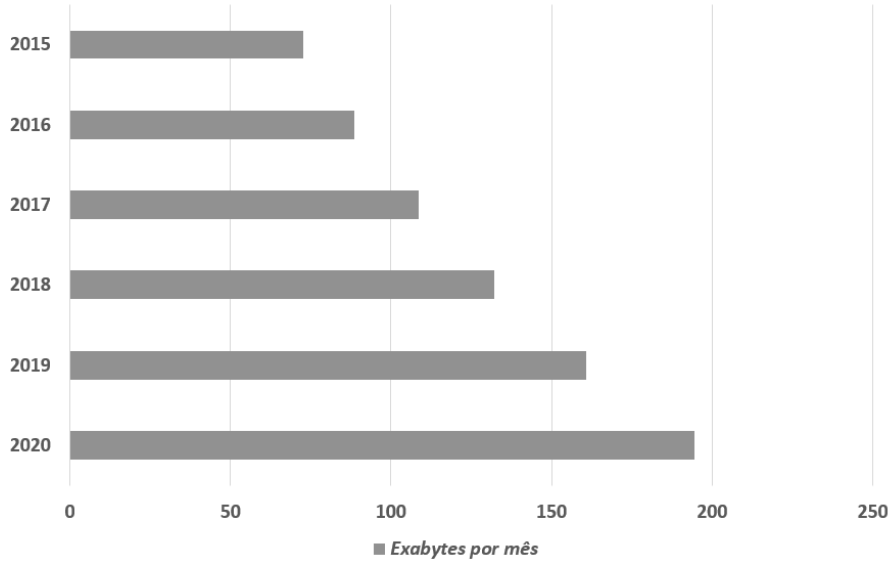
Neste capítulo são apresentadas as motivações que estimularam este trabalho, bem como o cenário que se enquadra a solução proposta, a problemática explorada, os objetivos deste trabalho e a proposta de metodologia de atuação, juntamente com a organização do trabalho.

1.1. Cenário

Com a previsão do tráfego IP global anual vir a ultrapassar o limite de 1 zettabyte (ZB), aproximadamente 1000 exabytes (EB), até o final de 2016 e em 2020, este chegar a 2,3 ZB por ano, como mostrado na Figura 1, e com uma previsão de cerca de 5 milhões de anos para assistir a quantidade de vídeos que atravessará as redes IP globais, a cada mês, em 2020, segundo divulgado pela empresa Cisco (2016), necessita-se de mecanismos cada vez mais precisos para formação de gostos, sugestões com base em perfis e até mesmo de buscas específicas feitas na rede.

Com a ampliação de opções se tem a vantagem do maior leque de vídeos, músicas, produtos e outros elementos disponíveis, que se mostraram compatíveis com as expectativas das sociedades. Porém, a capacidade de decisão frente a detecção de padrões, previsões, aprendizado e outros, dos softwares computacionais atuais será comprometida frente a um cenário composto por múltiplas e ótimas soluções possíveis.

Figura 1: Previsão para o tráfego em rede.



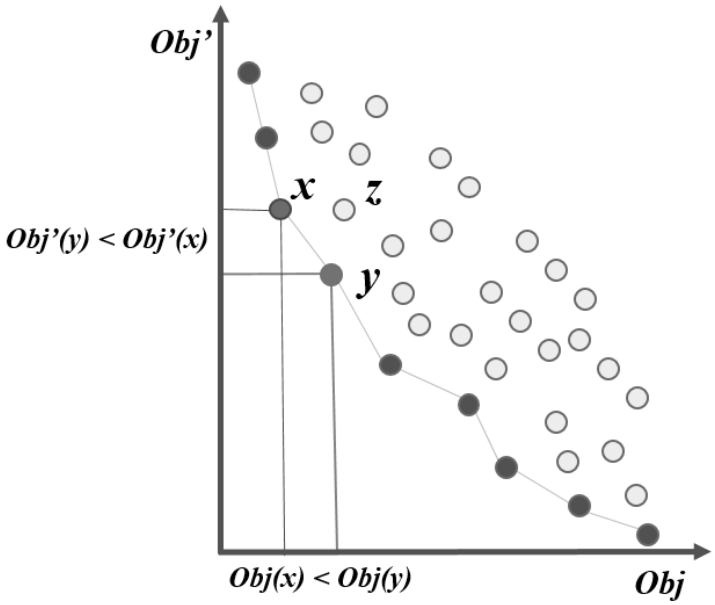
Fonte: Adaptado de Cisco (2016).

Essa variedade de soluções acaba, muitas vezes, caracterizando uma problemática multiobjetiva, pelo fato da tomada de decisão ter que atender a uma quantidade múltipla de critérios. Uma otimização multiobjetiva não é trivial, uma vez que à medida que o número de objetivos aumenta, torna-se mais provável que os empenhos se tornem complexos para atender a todos os critérios e, consequentemente, mais abstrusos de serem quantificados e devidamente atendidos. Para esses problemas mais complexos pode não existir uma única solução, que otimize, simultaneamente, cada objetivo. Nesse caso, as funções objetivas são ditas conflitantes, e existe um número de soluções ótimas de Pareto, como exemplificado na Figura 2.

A frente de Pareto, nome que faz alusão a Vilfredo Pareto (1848-1923), engenheiro e economista italiano, que usou o conceito, em seus estudos nos campos de economia e renda, traz um conjunto de soluções não dominadas, ou seja, nenhum objetivo pode ser melhorado sem sacrificar pelo

menos um dos demais objetivos. Por outro lado, uma solução é referida como dominada por outra solução se, e somente se, a solução dita dominante for igualmente boa ou melhor que a solução dominada em relação a todos os objetivos, sendo escolhida como ótima, como pode ser visto entre os elementos *x* e *y* dominantes e o dominado *z* na Figura 2 abaixo.

Figura 2: Demonstração de problema de minimização em frente de Pareto multiobjetiva.



Fonte: Próprio autor (2017).

A Inteligência Artificial (IA) pode ser utilizada como um meio de suporte para esses softwares, que requerem autonomia e robustez. Além de ser uma forte arma para propiciar a conceituação multiobjetiva, dinamismo nas mudanças e na heterogeneidade. Os agentes racionais como ferramenta de uso da Inteligência Artificial (IA), por sua vez podem ser utilizados como programas capazes de oferecer uma solução confiável para problemas com alto grau de complexidade e mutação.

1.2. Problemática

Tomando que um problema computacional é uma tarefa que, em princípio, pode ser resolvida pela aplicação de etapas matemáticas, este pode se tornar particularmente difícil de ser soluto à medida que maiores recursos sejam necessários para resolvê-lo. Porém, com recursos de processamento e armazenamento limitados nos aparelhos pessoais, estes requerem que existam cada vez maiores aprimoramentos nas sequências autossuficientes

algorítmicas empregadas para a solução dos problemas computacionais complexos advindos da era moderna.

Nas áreas que competem à computação ocorre o emprego da Inteligência Artificial (IA) como alternativa para atuar com a diversidade na complexidade de soluções, heterogeneidade e objetivos conflitantes. São várias as contribuições dadas pela área como as citadas por Russell e Norvig (2014), que afirmam que intérpretes interativos, ambientes de desenvolvimento rápido, estrutura de dados de lista vinculada, programação funcional, entre outros, foram originalmente desenvolvidos em laboratórios de Inteligência Artificial (IA).

O uso de agentes racionais como mecanismo de utilização da Inteligência Artificial (IA) vem provocando a criação de novas soluções dotadas de argúcia, que facilitam o modo de lidar com a complexidade dos problemas modernos. No entanto, como é de se esperar, um modo de lidar com problemas de grande complexidade também deve ser dotado de alguma complexidade. Nos agentes racionais, essa complexidade não está na criação dos programas agentes, pois já são diversas as ferramentas de auxílio para esta concepção, mas a natureza autônoma pode desafiar o entendimento de não peritos da área e intimidar acerca de uso desta, por receio baseado no desconhecimento de suas vantagens.

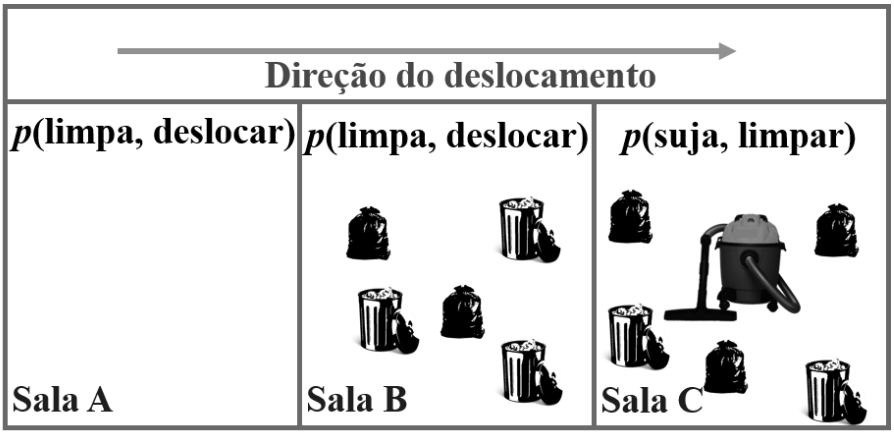
Para o projetista que possui afinco com o uso da Inteligência Artificial (IA), a aplicação de agentes racionais na solução de problemas é um passo corriqueiro e de grande eficácia, porém a garantia de correto funcionamento de um agente inteligente pode levar a passos exaustivos, tendo como base a ausência de ferramentas que facilitem essa fase de teste e a garantia de cumprimento do que se propõe.

A dificuldade em lidar com o teste de agentes racionais ocorre, principalmente, pela característica de autonomia na tomada de decisões e de seleção de ações, em que há a possibilidade da ação vista como melhor a ser feita pela visão do projetista não seja a mesma tomada como melhor opção pelo programa, em que ainda por vezes o agente acaba por encontrar uma solução não detectada por seu projetista, que supera a qualidade da ação esperada. E, ainda, em caso de falha do programa, ele pode deixar de tomar a ação correta pelo mau funcionamento de algum de seus componentes internos, não necessariamente ligados, de modo direto, ao conjunto de instruções algorítmicas de tomada de decisão.

Na Figura 3 é possível ver um exemplo, em que se supõe que um

agente de limpeza aspirador, que atua com base em reações simples para percepção, deixou de cumprir a função com eficiência por déficit de funcionamento em seu módulo de visão do ambiente de atuação, deixando para trás uma sala não aspirada, cuja sujeira não foi detectada, em que a função p(x, y) representa o conjunto x, y formado, respectivamente, por visão e ação. Nesse exemplo simples, a falha implica a baixa complexidade de ser notada, porém a causa da decisão incorreta apenas pode ser averiguada com uma verificação mais profunda de seus componentes e estados das variáveis essenciais em cada instante da atuação.

Figura 3: Atuação de um agente aspirador de pó.



Fonte: Próprio autor (2017).

São raras as abordagens que fazem alusão à criação de ferramentas de auxílio ao teste de agentes racionais, principalmente, dando liberdade ao projetista de usar a organização de arquitetura, que melhor lhe parecer. A primeira vista, o teste de um software dotado de autonomia pode não parecer tão abstruso, mas a real complexidade é extremamente desafiadora, pois desde a detecção até o ponto de origem da falha podem existir passos extenuantes, tendo em vista que em problemas de grande nível de dificuldade podem existir soluções de número não finito, e a conferência das saídas dadas pelos softwares autônomos pode se tornar um problema de proporções anômalas, caso o projetista não tenha a sua disposição mecanismos auxiliares.

Porém, embora haja dificuldades na fase de teste de softwares racionais, essa é uma fase que deve ser essencialmente cumprida, pois devem existir garantias de que o software atenda aos requisitos que orientaram sua concepção e desenvolvimento, e que este pode ser executado

em ambientes pretendidos, além de garantir as propriedades de usabilidade, escalabilidade, manutenção, dentre outras. Portanto, para facilitar a tarefa do projetista é necessária a concepção de novos mecanismos que auxiliem o teste de agentes racionais, tornando essa uma fase mais eficiente e eficaz e com menor grau de exaustão para o projetista.

1.3. Objetivos

Tendo em vista a ampliação do uso de agentes racionais, a expansão da Inteligência Artificial (IA) no cenário contemporâneo da Tecnologia da Informação, bem como diante da importância em oferecer soluções de software confiáveis, surge dentre os desenvolvedores a necessidade de recursos que facilitem o teste de agentes racionais, preenchendo essa lacuna de maneira a elevar ao máximo seu desempenho, maximizar a produtividade do projetista de software, bem como o tempo de concepção de tais softwares inteligentes. Neste contexto, o objetivo deste trabalho é contribuir com o teste de programas agentes racionais, por meio da concepção da proposta de um agente de monitoramento (*ProMon*), capaz de monitorar o desempenho do agente testado na execução de atividade em diferentes cenários de teste e diagnosticar as falhas apresentadas por este, gerando para o projetista informações relevantes sobre o desempenho do agente.

Para maximizar a efetividade do teste, esta proposta visa avaliação aprofundada dos subsistemas que compõem o agente testado (explorado mais adiante), por meio da conferência do estado interno das variáveis vitais das funções da estrutura interna do agente, de modo a detectar quais estão causando falhas. Com essa visão, é possível enviar ao projetista um relatório sobre o funcionamento dos subsistemas funcionais que compõem o agente, possibilitando a realização de melhorias substanciais nos programas agentes em tempo reduzido e com esforço mínimo na aplicação de teste.

A proposta também visa manutenção da heterogeneidade, de modo que possa lidar com a implementação de programas agentes nos mais diversos cenários e para a solução dos mais vastos problemas computacionais. De mesma forma é projetada de modo a dar a maior flexibilidade ao projetista, para que esse não seja limitado por padrões de arquitetura pouco flexíveis ou por um conjunto de restrições denso a ser seguido para que a ferramenta de auxílio ao teste lhe possa ser funcional.

No contexto de maximização do desempenho do agente testado,

ProMon contará com um módulo para a detecção de estados ideais para garantir que nenhuma ação do agente deixe de ser avaliada. Tal módulo é o que faz com que *ProMon* se torne capaz de atuar em ambientes diversos e atender problemas heterogêneos, para que possa acompanhar o projetista não somente em um único projeto de um problema específico, mas em qualquer projeto que faça uso da tecnologia de agentes racionais, propiciando o reaproveitamento de código e evitando retrabalhos, gerando ainda mais economia de tempo e ampliando a confiabilidade por meio dos testes.

1.4. Metodologia Proposta

Para que se alcancem os objetivos da proposta de forma interina, se faz necessária uma gama de apresentações de estudos e atribuições acerca do estado da arte do teste de agentes racionais, bem como uma modelagem da estrutura de *ProMon* para o entendimento das características que o envolvem. Por fins pedagógicos, diagramas são traçados para a orientação da fase de implementação da proposta e pseudocódigos de algumas funcionalidades essenciais são apresentados.

Na promoção de esclarecimentos acerca dos fluxos contidos em *ProMon*, diagramas de acompanhamento são introduzidos, também é apresentada a descrição e detalhamento das funcionalidades e importância de cada módulo projetado para *ProMon*, com suas respectivas peculiaridades.

Visando o melhor funcionamento de *ProMon* também são apresentadas técnicas para a avaliação de desempenho deste, para que se possa fazer toda e qualquer melhoria necessária, a fim de tornar o monitor o mais eficiente e eficaz possível, pois o sucesso do teste dependerá da perfeita atuação de *ProMon* frente ao agente testado. Alvitres de modos mais amigáveis de uso da proposta para quem possua pouca experiência em Inteligência Artificial (IA) também são apresentados, principalmente, no que diz respeito à utilização da proposta e interpretação de resultados por esta obtidos.

Portanto, o trabalho está organizado da seguinte forma: a seção dois apresenta um referencial teórico para auxiliar na compreensão do tema proposto na abordagem, funcionando como uma prévia de fundamentação teórica; na seção três são apresentados os trabalhos relacionados que serviram de base para a criação desta proposta; na seção quatro é apresentada a abordagem proposta, com os principais aspectos,

características da abordagem e descrições relacionadas; na seção cinco é apresentada uma proposta avaliativa, em que é ressaltada a descrição da experimentação e contextualização dos resultados esperados e da discussão. E, por fim, a conclusão e as pretensões de trabalhos futuros são apresentadas na seção seis.

2. Inteligência Artificial

Tomando-se a definição dada por Rich e Knigh (1994) de que a Inteligência Artificial (IA) é o estudo de como fazer computadores executarem coisas que, no momento, as pessoas fazem melhor, pois a conceituação é vasta e bastante discutida no meio acadêmico. Ainda, para conceituação se tem que Russell e Norvig (2014) dividem as numerosas definições de Inteligência Artificial (IA) em quatro categorias: Sistemas que pensam como seres humanos, Sistemas que agem como seres humanos, Sistemas que pensam racionalmente, Sistemas que atuam racionalmente.

A vastidão nas definições de Inteligência Artificial (IA) se apresenta como um indicativo do grau de complexidade. A Inteligência Artificial (IA) interessantemente emprega, em vastidão, o conceito de que certas características do universo e dos seres vivos são melhor explicadas por uma causa inteligente, não um processo não direcionado, onde podem haver cenários cuja complexidade seja irredutível e bem especificada, dando gancho para a busca da forma como estas causas foram projetadas e sua aplicação no meio da inteligência computacional. É utilizadora de inspiração advinda do meio natural para realização de tarefas como, por exemplo, no uso de algoritmos que têm a origem baseada em teorias acerca da evolução humana e do comportamento de animais, dentre outros.

2.1. Contexto Histórico

Mccorduck (2004) escreve que a Inteligência Artificial (IA) começou na antiguidade com um antigo desejo de forjar os deuses na criação de seres dotados de inteligência, rodeada de mitos e especulações, ainda complementa que a inteligência artificial de uma forma ou de outra é uma ideia, que permeou a história intelectual ocidental, um sonho que precisaria de realização urgentemente e saída do campo de lendas e de histórias. Por toda a história primordial, a Inteligência Artificial (IA) se baseou na suposição de que o processo do pensamento humano pode ser mecanizado, de forma a conseguir munir agentes robóticos de poder de raciocínio e de discernimento.

Os primeiros computadores modernos foram as enormes máquinas de quebra de código da Segunda Guerra Mundial como o ENIAC. Sendo que as

duas últimas destas máquinas foram baseadas no fundamento teórico lançado por Alan Turing (RUSSELL e NORVIG, 2014) e desenvolvido por John von Neumann (MCCORDUCK, 2004), passo evolutivo que tornou mais propício à criação sobre especulações da formalização do pensamento e inteligência humana.

Em 1950, Alan Turing sugeriu um teste para avaliação da habilidade de uma máquina em exibir um comportamento inteligente equivalente ou indistinguível do comportamento que um ser humano teria. Turing propôs que um avaliador humano julgasse as conversas de linguagem natural entre um ser humano e uma máquina especial, especificamente arquitetada para gerar respostas humanas. O avaliador estaria ciente de que um dos dois parceiros na conversa seria uma máquina, e todos os participantes seriam separados uns dos outros. A conversa ocorreria limitada a um canal de comunicação com suporte somente a texto, como um teclado e uma tela de computador, para que o resultado não dependesse da habilidade da máquina de transformação de palavras em linguagem sonora semelhante à humana. Se o avaliador não puder identificar a máquina e o humano entende-se que a máquina passou no teste. O teste não possui o intuito de verificar a capacidade de dar respostas corretas às perguntas, apenas o quão próximo se assemelham as respostas que um ser humano daria (TURING, 1950).

O campo da pesquisa da Inteligência Artificial (IA) foi cunhado em uma conferência no Dartmouth College, em 1956 e, atualmente, engloba uma enorme variedade de subcampos que vão desde os mais gerais, como aprendizagem e percepção até as mais específicas e peculiares tarefas, sendo que a Inteligência Artificial (IA) não tenta apenas compreender, mas também construir entidades inteligentes (RUSSELL e NORVIG, 2014).

Atualmente, a atuação da Inteligência Artificial (IA) está presente em diversos trabalhos e variados campos de pesquisa, como: em diagnóstico médico, veículos autônomos, criando arte como músicas, desafiando jogadores a vencê-los nos mais variados jogos, provando teoremas matemáticos, motores de busca, assistentes on-line, reconhecimento de imagens em fotografias dentre outros (RUSSELL e NORVIG, 2014). Ainda não há registro de uma Inteligência Artificial (IA) montada aos moldes da ficção, englobando robótica para uma fiel cópia humana, mas os avanços na mimetização dos movimentos humanos por robôs não ficam atrás dos atingidos pela Inteligência Artificial (IA).

2.2. Campo de Atuação

A Inteligência Artificial (IA) como ferramenta facilitadora das atividades do dia a dia está tão envolvida que para muitos pode até mesmo se passar por desapercebida, nas atividades comuns como no envio de um e-mail receber sugestões de contatos antes mesmo da digitação de um nome e de sua conclusão, correções de palavras digitadas nos smartphones ou mesmo a substituição da escrita pelos comandos de voz. Essas atividades oriundas da era tecnológica se tornaram tão naturais que aos poucos as pessoas são levadas a necessidade de cada vez ter maior praticidade e agilidade de tarefas, o que faz com que se deixe de fazer inúmeros questionamentos acerca do que realmente se está utilizando por trás das interfaces amigáveis e convidativas.

O mercado consumidor e as vertentes capitalistas sempre estão presentes em meio a demandas tecnológicas e a disseminação de uma nova tecnologia, afinal a inovação tecnológica tem passado a ser grande aliada na formação de novos negócios, tais como: os bancos virtuais como Nubank e o Banco Original, empresas que atuam com grande aceitação, mediante sites de avaliação em território nacional. Nesse contexto, a necessidade de cuidados com a saúde tem se tornado um mercado atraente para diversas empresas, o que já originou produtos que variam desde aplicativos para monitoramento do sono e braceletes de coleta de dados até mecanismos inteligentes de auxílio para tomada de decisão com os vindos da Watson Health da IBM.

O que pode, muitas vezes, não ser notado e que tamanhas evoluções não apenas nos cenários de saúde e finanças têm o envolvimento direto na Inteligência Artificial (IA), seja para a tomada de decisões, seja para lidar com a presença de um volume maciço de dados ou para permitir a aprendizagem de máquina. Vendo que o mundo caminha para uma evolução natural da tecnologia, e para uma maior necessidade de comodismo se pode afirmar que a Inteligência Artificial (IA) terá papel fundamental para possibilitar o mundo mais conectado e mais ágil frente aos desafios, que ainda estão por vir.

Propostas de soluções para problemáticas atuais, baseadas em Inteligência Artificial (IA), se alastram pelo meio acadêmico, porém para o projetista que busca o pleno desenvolvimento de solução e uma implantação de sucesso, ainda existem desafios a serem ultrapassados, alguns da própria natureza complexa da Inteligência Artificial (IA), outros vindos até da resistência pela formação de uma má visão do futuro auxiliado pela inteligência

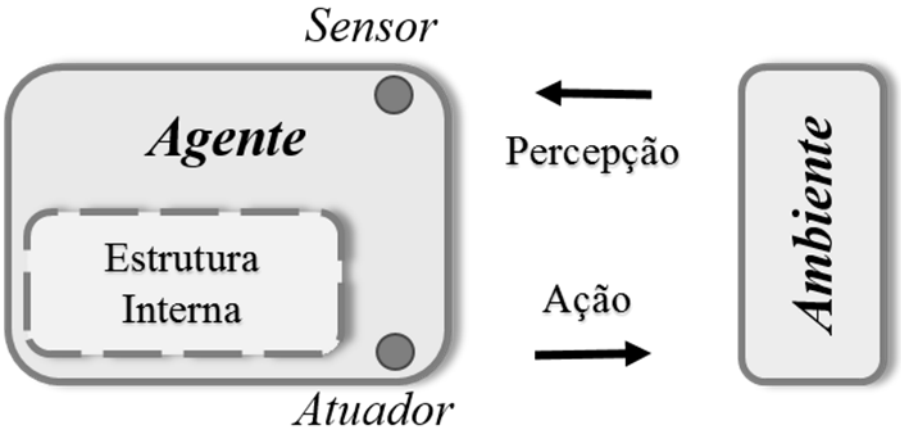
em softwares apoiadas por correntes de pensamentos filosóficos, que dizem respeito à nocividade de uma “máquina inteligente”.

Contudo, não se pode afirmar até onde vão os limites da Inteligência Artificial (IA), pois não é preciso decidir se uma máquina pode realmente "pensar", mas sim é necessário apenas decidir se uma máquina pode agir tão inteligentemente quanto um ser humano, para que possa solucionar, com maestria, o problema ao qual se propõe e elevar os níveis de satisfação.

2.3. Agentes Racionais

Não há definição universalmente aceita do termo agente, mas há um consenso geral de que autonomia é a ideia central envolvendo os vários conceitos (WOOLDRIDGE,1999). Portanto, toma-se que um agente é qualquer coisa que pode perceber o ambiente por meio de sensores e agir nesse ambiente por meio de atuadores, como pode ser conferida na estrutura demonstrada pela Figura 4.

Figura 4: Arquitetura básica de um agente.



Fonte: Adaptado de Russell e Norvig (2014).

Para que um agente possa atuar, em uma problemática, é necessário que este tenha uma função agente, que mapeia qualquer dada sequência de percepções para uma ação, que por sua vez será implementada por um programa de agente, em que a função agente é uma descrição matemática abstrata e o programa do agente é uma implementação concreta, rodando na arquitetura do agente.

Em Inteligência Artificial (IA) um agente racional pode ser

caracterizado por sua forma de atuação, em que para cada sequência de percepções possível, um agente racional deve selecionar uma ação que ele espera que venha a maximizar a medida de desempenho, dada a evidência fornecida pela sequência de percepções e por qualquer conhecimento interno do agente. Um agente racional é definido como aquele que age de forma a alcançar o melhor resultado ou, quando em incerteza, o melhor resultado esperado (RUSSELL e NORVIG, 2014). A medida do desempenho de um agente deve refletir o resultado realmente desejado.

Wooldridge (1999) visualiza um agente como sendo uma entidade com capacidade de resolução de problemas encapsulada. Inserido nesta visão, define o agente como tendo as seguintes propriedades:

- **Autonomia:** Executa a maior parte das ações sem interferência direta de agentes humanos ou de outros agentes computacionais, possuindo controle total sobre as ações e estado interno. Para ter autonomia, o agente deve ter certo grau de inteligência, capacitando-o a sobreviver em um ambiente dinâmico e, por vezes, não benigno.

- **Habilidade social:** Por impossibilidade de resolução de certos problemas ou por outro tipo de conveniência interagem com outros agentes (humanos ou computacionais), para completarem a resolução de seus problemas ou, ainda, para auxiliarem outros agentes. Disto surge a necessidade de que os agentes tenham capacidade para comunicar os requisitos aos outros e um mecanismo decisório interno, que defina quando e quais interações são apropriadas.

- **Reatividade:** Percebem e reagem às alterações no ambiente (que pode ser o mundo real, a internet, uma mistura de mundos, entre outros) em que estiverem inseridos.

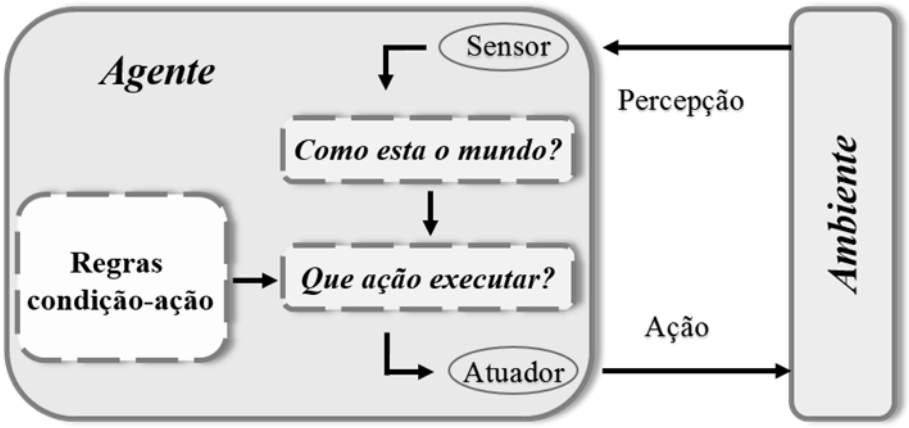
- **Proatividade:** Agentes, do tipo deliberativo, além de atuarem em resposta às alterações ocorridas em seu ambiente, apresentam um comportamento orientado para objetivos, tomando iniciativas, quando julgarem apropriado.

Russell e Norvig (2004) dividem os sistemas agentes em quatro tipos principais, sendo estes programas agentes: (i) agentes reativos simples, (ii) agentes reativos baseados em modelos, (iii) agentes baseados em objetivos e (iv) agentes baseados em utilidade.

Agentes reativos simples selecionam ações com base somente na percepção atual ignorando o resto da história de percepções. O funcionamento

do agente reativo é baseado em regras de condição-ação: se condição faça ação (Figura 5). São simples, porém, limitados, pois funcionarão somente se a decisão correta puder ser tomada com base apenas na percepção atual e forem utilizados em condições de ambiente completamente observável. O algoritmo do agente reativo simples é mostrado na Figura 6.

Figura 5: Arquitetura básica de uma agente reativo simples.



Fonte: Adaptado de Russell e Norvig (2014).

Figura 6: Algoritmo de um agente reativo simples.

```
programa
{
 funcao AgenteReativoSimples (percepcao) retorna acao
          {
          estatico: regras, conjunto condicao-acao
          estado <= DecifrarEntrada(percepcao)
          regra <= ApropriacaoDeRegra(estado, regras)
          acao <= RegraDeAcao(regra)
          retorne acao
          }
}
```

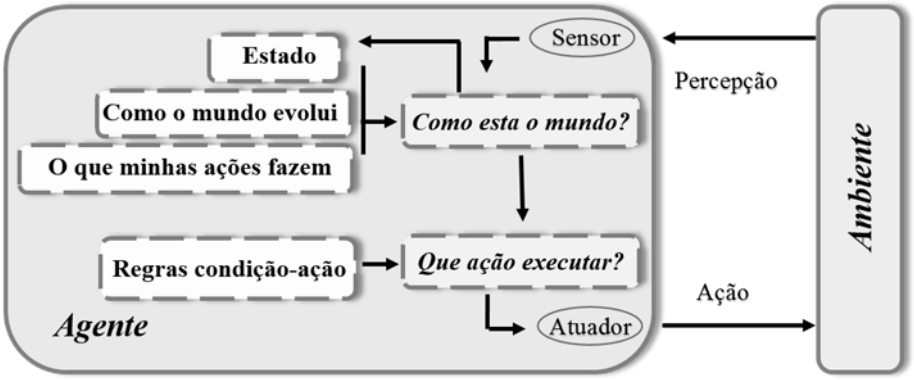
Fonte: Adaptado de Russell e Norvig (2014).

Um **agente reativo baseado em modelo** é capaz de lidar com ambientes parcialmente observáveis. O agente deve controlar as partes do mundo que ele não pode ver, deve manter um estado interno, que dependerá do histórico de percepções e reflita os aspectos não observados, no estado atual, como mostrado na Figura 7. Esse conhecimento sobre "como o mundo funciona" é chamado de modelo do mundo, daí o nome de "agente baseado

em modelos". Um agente baseado em modelo é um agente que usa um modelo de mundo. O programa do agente é mostrado na Figura 8.

Agentes baseados em objetivos expandem as capacidades dos agentes baseados em modelos, por meio de um “objetivo”. O objetivo descreve situações de otimalidade desejáveis, em que a seleção da ação baseada em objetivo pode ser feita, de forma direta, quando o resultado de uma única ação atinge o objetivo ou mais complexa quando serão necessárias longas sequências de ações para atingir o objetivo.

Figura 7: Arquitetura básica de uma agente reativo baseado em modelo.



Fonte: Adaptado de Russell e Norvig (2014).

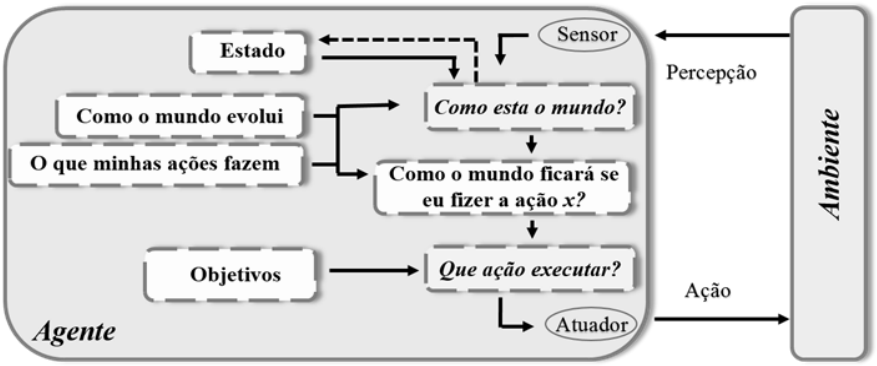
Figura 8: Algoritmo de um agente reativo baseado em modelo.

```
programa
{
 funcao AgenteReativoBaseadoEmModelo (percepcao) retorna acao
            {
            estatico: regras, estado, conjunto condicao-acao
            estado <= AtualizaEstado(estado, acao, percepcao)
            regra <= ApropriacaoDeRegra(estado, regras)
            acao <= RegraDeAcao(regra)
            retorne acao
            }
}
```

Fonte: Adaptado de Russell e Norvig (2014).

Para encontrar sequências de ações, que alcançam os objetivos, são utilizados algoritmos de Busca e Planejamento. A tomada de decisão envolve a consideração do futuro, o que não acontece com o uso de regras de condição – ação como mostrado na Figura 9.

Figura 9: Arquitetura básica de uma agente reativo baseado em objetivo.



Fonte: Adaptado de Russell e Norvig (2014).

O agente que funciona orientado a objetivos é mais flexível do que um agente reativo. Entretanto, o objetivo não garante o melhor comportamento para o agente, apenas a distinção entre estados objetivos e não objetivos.

Agentes baseados na utilidade buscam definir um grau de satisfação com os estados. O quanto o agente está próximo de seu objetivo com aquele estado (Figura 10). Se um estado do mundo é mais desejável que outro, então ele terá maior utilidade para o agente, sendo assim, a utilidade é uma função que mapeia um estado para um número real, que representa o grau de satisfação com este estado.

Ao projetar um agente, a primeira etapa deve ser sempre especificar as decididas configurações para o projeto do agente inteligente. Para melhor executar essa especificação perguntas podem ser feitas, a fim de contentar em resposta ao conjunto de características a que se chamará de PEAS (*Performance, Environment, Actuators, Sensors*), o que vale dizer que o problema que é soluto por um agente pode ser caracterizado por medidas de desempenho, de ambiente, de atuadores e de sensores.

Figura 10: Arquitetura básica de uma agente reativo baseado em utilidade.

Fonte: Adaptado de Russell e Norvig (2014).

As medidas de desempenho são responsáveis por determinar o grau de sucesso do agente, os atuadores são os mecanismos de concepção da ação quer sejam físicos ou não, os sensores são os responsáveis por capturar o conjunto de percepções do agente e o ambiente é o mundo no qual o agente agirá para prover soluções para as problemáticas a que se propõe.

Russell e Norvig (2004) sugerem uma classificação dos ambientes, com base em propriedades, sendo:

- **Totalmente observável / Parcialmente observável:** Refere-se à possibilidade de um agente obter informações completas e precisas sobre o estado do ambiente, um ambiente totalmente observável é aquele em que o agente tem acesso a todas as informações relevantes, no ambiente, para a tarefa.

- **Determinismo / não-determinismo:** Em um ambiente determinista, qualquer ação tem um único efeito garantido, e nenhuma falha ou incerteza. Um ambiente é determinístico se o próximo estado puder ser perfeitamente predito, dado o conhecimento do estado precedente e a ação do agente. Pelo contrário, é um ambiente não determinista. Em um ambiente não-determinístico, a mesma tarefa executada duas vezes pode produzir resultados diferentes ou mesmo falhar completamente.

- **Episódico / Sequencial:** Ambientes episódicos se mostram como uma série de ações one-shot, e apenas a percepção atual (ou recente) é relevante. Em um ambiente episódico, o desempenho de cada agente é o resultado de uma série de tarefas independentes realizadas. Não há nenhuma ligação entre o desempenho do agente e outros cenários diferentes. Em outras palavras, o agente decide qual ação é melhor tomar, ele só irá considerar a tarefa em mãos e não tem que considerar o efeito que pode ter em tarefas futuras. Já os ambientes sequenciais exigem memória de ações passadas para determinar a próxima melhor ação.

- **Estático / Dinâmico:** Um ambiente é estático se apenas as ações de um agente o modificarem, ou seja, os ambientes estáticos não mudam enquanto o agente delibera. O dinâmico, por outro lado, é visto se outros processos estão operando sobre ele.

- **Discreto / Contínuo:** Um ambiente é caracterizado como discreto se houver um número finito de ações, que podem ser executadas dentro

dele, ou seja, um ambiente discreto tem locais fixos ou intervalos de tempo. Um ambiente contínuo poderia ser medido, quantitativamente, para qualquer nível de precisão, mas não apresentando um número limitado de estados distintos, claramente definidos do ambiente como um discreto. Um ambiente pode ser contínuo no que diz respeito às percepções do agente e discreto no que diz respeito às ações (e vice-versa).

- **Agente único / Multiagentes:** Caso o ambiente contenha outros agentes, que podem ser do mesmo tipo ou de tipos diferentes é denominado a este que seja multiagente. Caso contrário, será um sistema centralizado com agente único.

Quanto mais complexo for um ambiente, mais difícil será decidir qual ação executar. O ambiente mais complexo seria aquele que é parcialmente observável, estocástico, sequencial, dinâmico, contínuo e multiagente. As dificuldades em trazer soluções ótimas, em meio aos processos de complexidade se apresentam refletidos, diretamente, tanto na concepção do agente propriamente dito como na fase de conferência da solução, por meio de testes tornando a tarefa do projetista mais custosa e desafiadora.

2.4. Testes de Softwares Dotados de Racionalidade

O teste de softwares é uma sistematização investigativa conduzida para fornecer às partes interessadas (projetistas, clientes, outros) informações sobre a qualidade do produto ou serviço sob teste. Pode ser tomado como um processo de validação e verificação de que um programa de software, aplicação ou produto atende requisitos que orientaram a concepção e o desenvolvimento, executando funções dentro de um tempo aceitável, sendo suficientemente utilizável, de forma que pode ser instalado e executado em seus ambientes pretendidos e atingir o resultado geral que as partes interessadas almejam.

O objetivo principal da fase de teste é detectar falhas de software para que os defeitos possam ser devidamente corrigidos. A fase de teste não pode estabelecer que um produto funcionasse, adequadamente, em todas as condições, mas pode estabelecer que existissem padrões de funcionamento inadequado em condições específicas. O teste pode não apenas gerar informações indicativas de pequenas correções, como também pode gerar informações, que podem ser usadas como molde para corrigir o processo pelo qual o software foi desenvolvido.

O teste de softwares autônomos, ou agentes racionais pode ser feito seguindo diversas metodologias já amplamente difundidas no meio computacional com as devidas adaptações para lidar com as características próprias dos agentes, porém sem atingir o máximo potencial. As técnicas que não somente podem ser aplicadas ao teste de softwares comuns, mas também ao teste de softwares inteligentes apresentam características diversas e devem ser avaliados com base em eficiência e eficácia como, por exemplo, pelos níveis a que são aplicados os testes, pois para estes pode haver abordagens que, por alguma falha ou parâmetro, sejam aplicados os testes somente em algumas das partes componentes do programa agente, deixando de gerar testes eficazes.

É também aconselhável que para o teste considerar adotar qualquer estrutura em testar o comportamento do agente, de modo que se possa maximizar os resultados e minimizar os gastos de tempo. Para essas estruturas vale optar por práticas comprovadamente eficazes, cuja construção tenha melhor organização e respeito para com as premissas, que moldam a construção de um software inteligente, e que não faça uso de ferramentas de auxílio, que sejam de cunho duvidoso, podendo limitar a ação da estrutura de teste.

Para compor meios de comprovação de funcionamento de ferramentas de teste pode-se atentar para a apresentação de sistematizações como um estudo de caso, que é um relato de uma atividade, evento ou problema que contém uma situação real ou hipotética, que amplia a visão sobre a correta construção da ferramenta. A aptidão de lidar com a capacidade de evolução de um agente, por parte de uma estrutura de testes também não pode ser menosprezada, pois a capacidade evolutiva faz referência à habilidade do agente de tomar as ações necessárias para se adaptar a possíveis novas exigências e ajustar-se de acordos com estas. Portanto, uma ferramenta pode dispor de mecanismos de geração e evolução de casos de teste, de forma automática, trazendo novas exigências e desafios ao agente sob teste.

As características de cobertura da técnica de teste de software empregada devem ser analisadas, para que se tenha plena convicção de que fornecem, devidamente, o suporte para a necessidade do projetista, que executará os testes. Outro aspecto que deve ser medido, previamente, é o tipo de parâmetro de entrada utilizado para realização dos testes, quer sejam baseados na especificação, no modelo ou na codificação, a fim de se garantir a disponibilidade desses para atingir, sem problema, o estado de teste ideal da estrutura de teste utilizada.

O teste quando direcionado mais precisamente para sistemas multiagentes consiste em cinco níveis, conforme proposto por Nguyen (2008) e Moreno, Pavón e Rosete (2009), sendo esses níveis: de unidade, de agente, de integração, de sistema e de aceitação. Sendo que cada nível possui o leque de objetivos, assuntos a serem tratados no teste e atividades correspondentes:

- **Unidade:** Teste que abrange todas as unidades que compõem um agente, incluindo blocos de código, implementação das unidades particularidades do agente como as responsáveis por metas, planos, base de conhecimento, módulo de raciocínio, especificação de regras e, assim, por diante, com o intuito de se certificar de que estas funcionam exatamente como projetado.
- **Agente:** Teste de integração dos diferentes módulos dentro de um agente; testa a capacidade dos agentes no cumprimento dos seus objetivos e na capacidade de sentir seu ambiente de atuação.
- **Integração:** Testa a interação de agentes, protocolos de comunicação e semântica, interação dos agentes com o ambiente de atuação, de integração dos agentes com recursos compartilhados, que faz a aplicação de regulamentos, a observação de propriedades emergentes e comportamentos coletivos, buscando que um grupo de agentes, de recursos do ambiente funcionem corretamente juntos.
- **Sistema:** Testa um sistema multiagente em execução no seu ambiente operacional de destino, visa o teste das propriedades emergentes e macroscópicas esperadas do sistema, como um todo, além das propriedades de qualidade pretendidas que o sistema deve alcançar, tais como: adaptação, abertura, tolerância a falhas e desempenho.
- **Aceitação:** Executa o teste de um sistema multiagentes no ambiente de execução de demanda do cliente e verifica se ele atende aos objetivos almejados pelas partes interessadas, contando com a participação direta das partes interessadas para aferir sobre a avaliação de teste.

Levando em conta que a execução de testes de software está fortemente relacionada com a qualidade final do produto desenvolvido, os testes são uma das alternativas principais que uma equipe de desenvolvimento possui para obter um produto com um número reduzido de defeitos, já que a total correção de um software pode se tornar uma visão

idealizada com muitas dependências e variáveis. A automação desta fase de testes pode trazer benefícios diretos, que atingem a característica de repetitividade e custo de tempo e, em melhores casos, até a ampliar o poder desta fase por análises autônomas mais aprofundadas, que para o projetista são de grande valia no aumento substancial da produtividade, da qualidade e de aceitação de produtos concebidos.

3. Trabalhos Relacionados

Uma visão detalhada das pesquisas realizadas se faz necessário para identificar as lacunas, cujas contribuições deste trabalho se encaixam. Nesse sentido, serão apontados alguns trabalhos no contexto de teste de agentes racionais, com as respectivas características e análises, conforme identificados a seguir.

3.1. Viabilidade de Teste de Sistemas de Agentes BDI

Winikoff e Cranefield (2014) exploram a intuição de que sistemas agentes são difíceis de testar e analisam os possíveis comportamentos de agentes *Belief-Desire-Intention* (BDI), isto é, o número de caminhos por meio de um programa BDI e a probabilidade de falha. É utilizado na proposta o acompanhamento de diversos parâmetros, como o número de planos aplicáveis por meta e a origem de falha, além de constatar as dificuldades no teste de agentes, também trazem análises sobre a influência dos parâmetros no número de possíveis caminhos a seguir, em agentes BDI.

Os autores priorizaram uma visualização da execução do BDI não como um processo, mas como uma transformação de dados de uma árvore de plano de meta (finita) em uma sequência de ação - execução. Os eventos e planos são representados visualmente, como uma árvore, em que cada meta tem como folhas instâncias de planos que são aplicáveis, e cada instância de plano tem como folhas os sub-objetivos que são publicados pelos planos (Figura 11). A visualização da execução do BDI em termos de uma árvore de planos de metas e de sequências de análise do tamanho do espaço de comportamento tem por objetivo facilitar a análise para consolidar a proposta.

Figura 11: Gramática para a representação em árvore do plano de meta.

$$
\begin{array}{rcl}
\langle GPT \rangle & ::= & \texttt{goal([])} \mid \texttt{goal([}\langle PlanList \rangle\texttt{])} \\
\langle PlanList \rangle & ::= & \langle Plan \rangle \mid \langle Plan \rangle\texttt{,}\langle PlanList \rangle \\
\langle Plan \rangle & ::= & \texttt{plan([])} \mid \texttt{plan([}\langle AoGL \rangle\texttt{])} \\
\langle AoGL \rangle & ::= & \texttt{act(}A\texttt{)} \mid \langle GPT \rangle \mid \texttt{act(}A\texttt{),}\langle AoGL \rangle \mid \langle GPT \rangle\texttt{,}\langle AoGL \rangle
\end{array}
$$

Fonte: WINIKOFF; CRANEFIELD, (2014).

Pela característica de centrarem no modelo BDI, principalmente, com foco em espaço comportamental, os autores identificam diversos fatores, que influenciam o tamanho deste espaço comportamental. A questão por eles tratada diz respeito a quantidade de possibilidades comportamentais, de ações aceitáveis como soluções para os problemas aplicáveis a agentes BDI, cuja forma de solução foi traçada derivando fórmulas, que permitem calcular o número de comportamentos bem-sucedidos e mal sucedidos (falhas) para uma determinada árvore meta-plano.

Contudo, os autores concluem que o número de possíveis condutas para um agente BDI cresce à medida que a profundidade e a amplitude da árvore crescem, mas se surpreendem ao notarem que a introdução de tratamento de falhas traz uma diferença muito significativa para o número de comportamentos. E, embora, eles tenham grande trabalho, acabam por terem resultados que sugerem fortemente que não é viável o teste de todo o sistema devido a grande quantidade de possibilidades de soluções a ser avaliada, o que tornaria o processo muito exaustivo e sem um custo benefício computacional viável. Com a conclusão que o teste de um sistema BDI todo não é viável, os autores sugerem possíveis abordagens para lidar com essa questão da capacidade de teste, com o intuito de encorajamento de tentativas de terceiros.

3.2. Teste Evolutivo de Agentes

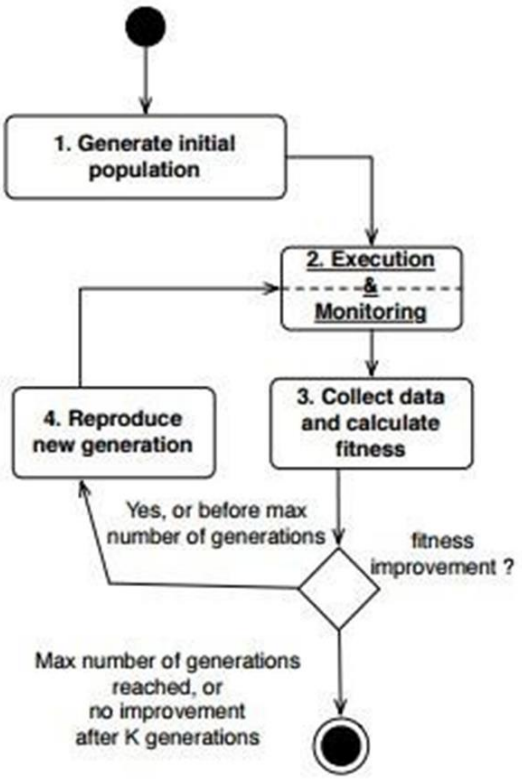
Nguyen et al. (2009) aborda o teste de agentes considerando a natureza deste, levando em conta que os agentes autônomos podem reagir de maneiras diferentes para as mesmas entradas ao longo do tempo e isso juntamente com mudança de influência de fatores pode propiciar falhas na atuação ao longo do tempo. Para sanar o problema, os autores aplicam uma sistematização que introduz e avalia uma abordagem para testar agentes autônomos que usam a otimização evolutiva para gerar casos de teste cada vez mais exigentes.

A abordagem do problema de geração de casos de teste desse trabalho faz uso das funções de qualidade, derivadas de requisitos relacionados com a autonomia, como mecanismo de avaliação do comportamento dos agentes autônomos. Sendo o principal objetivo do trabalho avaliar o desempenho de agentes autônomos de forma automatizada, fazendo uso de algoritmos evolucionários para gerar casos de testes mais exigentes, buscando expor as falhas do agente. A Figura 12 mostra um fluxograma com os procedimentos que compõem o teste evolucionário.

A proposta apresentada busca derivar medidas de aptidão a partir das exigências das partes interessadas e usá-las como critérios de avaliação para agentes de software. Durante a execução dos testes, os agentes sob teste são monitorados, os valores de aptidão são então calculados com base nos dados do monitoramento e utilizados para avaliar os agentes, com o intuito de gerar os testes variados com níveis crescentes de dificuldade.

A proposta busca uma forma sistemática avaliar a qualidade dos agentes autônomos. Como os requisitos das partes interessadas são representados como medidas de qualidade, os agentes autônomos têm de cumprir um limiar no âmbito destas medidas para que possam ser classificados como prontos para uso das partes interessadas. As funções de desempenho que são apresentadas como a representação dos objetivos do teste são definidas com base nas funções de qualidade e orientam a técnica de geração de testes evolutivos para gerar casos de teste automaticamente.

Figura 12: Procedimento de teste evolucionário.



Fonte: NGUYEN et al., (2009).

Os resultados obtidos a partir do conjunto de experiências realizadas pelos autores desse trabalho fornecem evidências para apoiar a alegação de que o teste evolutivo multiobjetivo pode gerar casos de testes, que satisfaçam

múltiplos propósitos de teste, criando condições múltiplas, nas quais falhas podem ser reveladas, considerando que quanto mais tempo estiver disponível para a evolução, mais desafiadores são os casos de teste evoluídos. Assim, o agente autônomo é testado, cada vez mais, extensivamente.

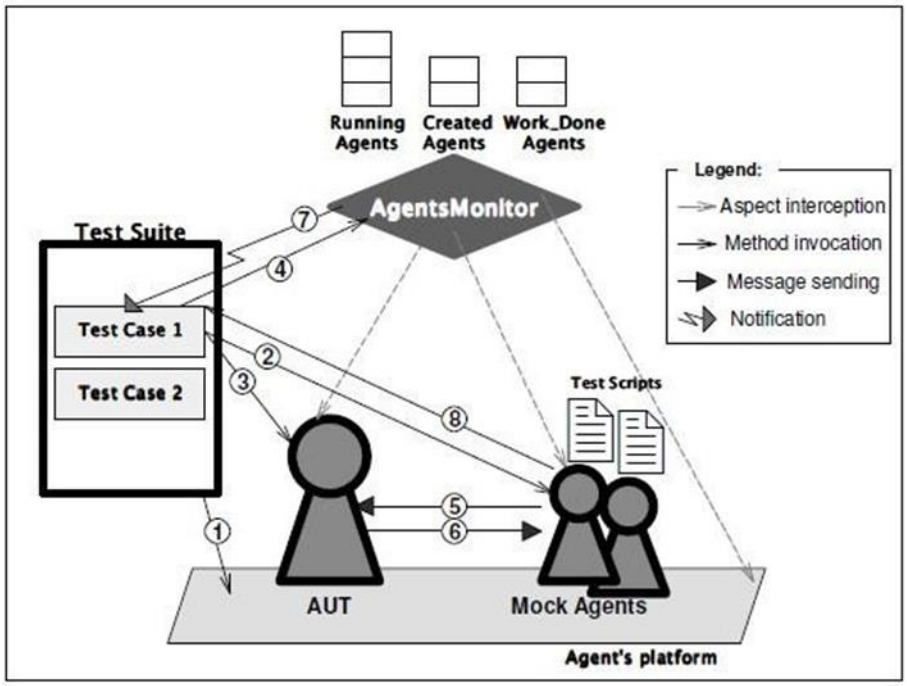
3.3. Teste Unitário em Sistemas Multiagentes baseado em Agentes Simulados

Coelho et al. (2006) apresentam uma abordagem auxiliada pela plataforma JADE (Java DEvelopment Framework) que é um software Framework que simplifica a implementação de sistemas multiagentes através de um middleware, com o principal objetivo de ajudar os desenvolvedores de agente com o teste individual em sistemas multiagentes para orientar o projeto e execução de casos de teste, por meio do monitoramento e controle da execução assíncrona dos agentes durante o teste. Para tal é explorada a ideia de que um sistema multiagente, em execução, é uma rede de agentes, que interagem uns com os outros e com um ambiente de atuação externo.

A abordagem de testes proposta chama a atenção para o teste dos menores blocos de construção que compõem os sistemas multiagentes: os próprios agentes. A ideia básica é verificar se cada agente, em isolamento, respeita as especificações tanto em condições normais de atuação (modos de atuação previstos) quanto em regimes anormais de saturação e estresse. Cada teste de unidade de agente segue a estrutura comum mostrada na Figura 13, onde AUT é o agente testado.

Em geral, o trabalho assume que a metodologia menos eficaz para a seleção de um subconjunto de todos os possíveis casos de teste é a escolha feita de forma arbitrária de um conjunto de casos de teste, pois em termos da probabilidade de detectar a maioria dos erros (estado desejável de teste), uma coleção arbitrariamente selecionada de casos de teste teria pouca chance de ser um subconjunto ótimo, ou mesmo próximo ao ótimo. E acabam por adotar na abordagem de ensaio unitário para os sistemas multiagentes, propondo uma metodologia para a seleção de casos de teste não definida pela literatura até então, apresentando assim uma ideia básica de uma técnica de adivinhação de erros, que consiste em enumerar uma lista de possíveis situações propensas a erros e, então, escrever casos de teste com base nessa lista criada.

Figura 13: Fluxo de trabalho entre os participantes de um teste unitário



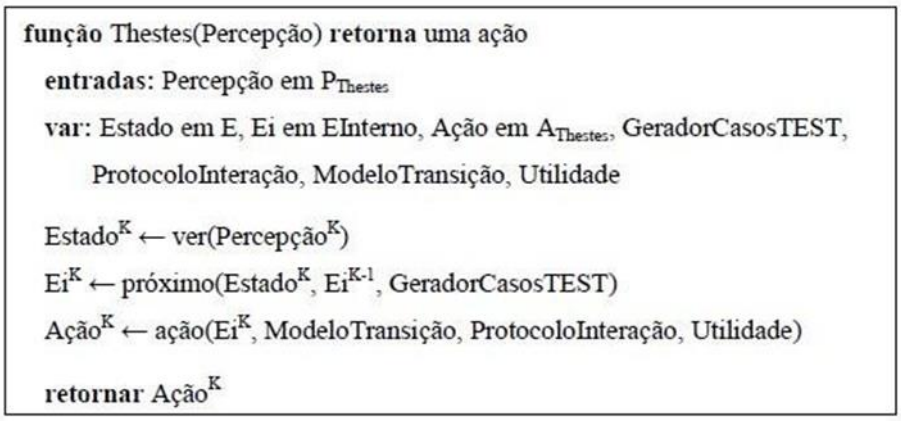
Fonte: COELHO et al., (2006).

No final, a abordagem tem por intuito ajudar os desenvolvedores de sistemas multiagentes a testar cada agente individualmente. Baseando-se no uso de agentes simulados para orientar o design e a implementação de casos de teste unitários de agentes. Cada agente simulador executa um script de teste, no qual envia e recebe mensagens do agente que está sendo testado, sendo o responsável por testar um único papel de um agente, em cenários em que tenha sucesso em falhas excepcionais.

3.4. Agentes Racionais para o Teste de Agentes Racionais

Silveira (2013) motivada pela lacuna existente em termos de técnicas de testes aplicados, de forma mais específica, para a avaliação do comportamento dos sistemas baseados em agentes, propõe uma abordagem baseada em um agente de resolução de problemas de seleção de casos de teste (Thestes). O esqueleto principal de Thestes (apresentado na Figura 14) fundamenta-se na estrutura de um programa agente orientado por utilidade. Durante o processo de busca de um conjunto de casos de teste, o agente Thestes utiliza um protocolo de interação entre o agente testado e seu ambiente para realizar simulações, tomar nota e avaliar, por meio de uma função de utilidade, as histórias correspondentes a essas simulações.

Figura 14: Esqueleto de Thestes.



Fonte: SILVEIRA, (2013).

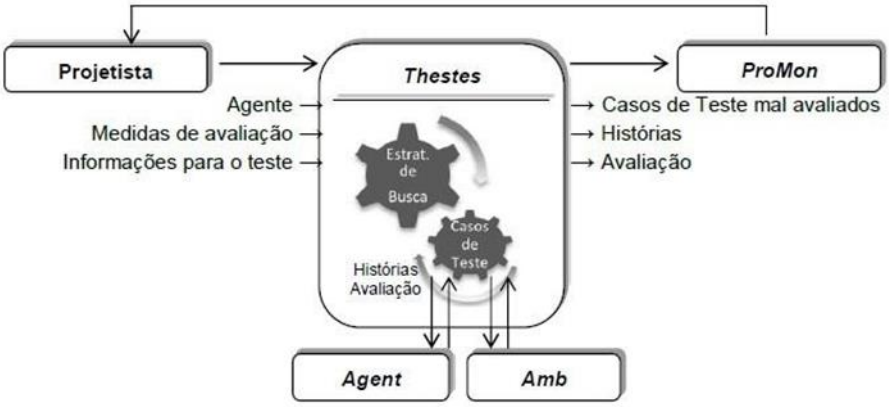
Silveira (2013) formula Thestes como um agente, que incide na resolução de problemas de seleção de casos de teste, que realiza busca local no espaço de estados de casos de teste orientado por utilidade. Foi concebido como um agente de resolução de problemas de seleção de caso de testes com estrutura do programa agente orientado por utilidade. O programa emprega uma estratégia de busca local, baseada em populações e orientada por uma função de utilidade para encontrar conjuntos de casos de teste satisfatórios, ou seja, ambientes específicos em que as histórias associadas de um agente testado em um dado ambiente de atuação que possuem baixo desempenho. Considerando que o processo de teste do agente dependerá dos casos de teste selecionados, Thestes objetiva selecionar um conjunto de casos de teste ideais, ou seja, aquele que, na simulação da avaliação de desempenho das interações do agente testado com seu ambiente (histórias de atuação), o agente não foi bem avaliado, o que consiste em obter um baixo desempenho.

Considerando o estado interno atualizado, a função ação de Thestes inicia um processo de busca local visando encontrar uma ação satisfatória. Essa função utiliza informações a respeito de um modelo de transição de estados para gerar novos casos de teste, além do protocolo de interação e uma função utilidade para, respectivamente, obter as histórias correspondentes aos casos de teste no conjunto e avaliar o desempenho dos agentes testados nessas histórias de teste. Assim, o conjunto de casos de testes iniciais e o modelo de transição permitem que a ação de Thestes gere o espaço de estados do problema de seleção, bem como a função utilidade permitem que se avaliem os estados gerados e, consequentemente, o comportamento do agente testado nesses estados.

Para que Thestes tenha plena funcionalidade, ele mantém o conjunto

o caso de teste e os episódios em que o agente teve o comportamento mais inadequado em todas as gerações, ou seja, os casos de teste cujo valor de utilidade representa o valor máximo entre todos os casos de teste avaliados anteriormente. O objetivo de Thestes é identificar situações, em que o agente testado possui o pior valor de avaliação.

Figura 15: Arquitetura proposta de Thestes.



Fonte: SILVEIRA, (2013).

Para a geração de casos de teste, a proposta apresentada por Silveira (2013) faz uso de algoritmos evolutivos, mais precisamente do NSGA-II e do algoritmo genético. Também idealiza o conceito de um agente de monitoramento (*ProMon*) capaz de avaliar os subsistemas que compõem um agente testado, a fim de encontrar a origem das falhas a partir dos casos de teste mal avaliados e de um conjunto de soluções ideais. A arquitetura da proposta pode ser vista na Figura 15 e está centrada no agente de seleção de casos de teste Thestes. A apresentação de *ProMon* será a origem do trabalho aqui apresentado.

As informações geradas pela abordagem indicam uma medida de utilidade média associada ao desempenho do agente testado e os objetivos na medida de avaliação que não estão sendo satisfeitos. Considerando o melhor conjunto de histórias do agente em seu ambiente, associado ao conjunto de casos de teste selecionados pela abordagem ao final do processo de busca, o projetista e/ou outros sistemas automáticos auxiliares podem identificar aqueles episódios problemáticos, e quais subsistemas de processamento da informação e módulos de informação associados estão causando o desempenho insatisfatório no agente auxiliando, assim, no teste de agentes racionais.

3.5. Análise e Considerações

Apresentados os trabalhos pode-se concluir que a área que abrange o teste de agentes autônomos é em demasiado complexa, pois a verificação de todos os comportamentos adotados por um agente pode ser extensivamente exaustiva e demandar muitos recursos computacionais. A propriedade de autonomia dos agentes é um fator desafiador, pois dentre o leque de possibilidades (que pode acontecer dentro de um conjunto de soluções não finitas) o agente pode optar por qualquer escolha, considerando seu objetivo.

Com conclusões que rodeiam os resultados dos trabalhos citados, é possível destacar como fechamento que a construção de uma ferramenta de teste genérica, que possa atender a todas as necessidades de um projetista de sistemas autônomos, que busque testes mais eficientes e eficazes, uma vez que ainda se está apenas no desejo dos pesquisadores da área. Embora grandes avanços já tenham sido feitos em áreas específicas de testes, a generalidade das soluções ainda é pouco explorada, o que juntamente com as derivações de abordagens de programas agentes acaba acarretando no não fechamento de um conjunto de sistematizações de testes, globalmente utilizadas, para softwares dotados de autonomia.

Dos desafios envolvidos na fase de teste, a precisão na indicação do ponto de falha do agente merece evidência, pois a indução do estresse de um sistema autônomo pela atuação em cenários de maior complexidade pode levá-lo a pontos falhos em sua execução, que podem ser identificados em técnicas comparativas, considerando que o projetista está munido com o conjunto de ações esperadas como ótimas, tabuladas previamente no escopo de atuação de possibilidades finitas de comportamentos que o agente testado atuou. Porém, a detecção de um ponto de falha pode não gerar caminhos simples para identificação de seu ponto de origem, o que gera para o projetista uma condição que pode ser frustrante, pois a correção da falha dependerá da precisão da localização e identificação da mesma.

As deficiências das propostas para o teste de agentes podem ser facilmente justificáveis pela exploração de um escopo finito, pois para o pesquisador a abordagem de uma porcentagem de características dos agentes autônomos confere maior foco e rendimento para a pesquisa. No entanto, a limitação demasiada pode empobrecer o trabalho, diminuindo o grau de impacto e de relevância para todas as partes interessadas. Por esses motivos, uma abordagem que compreende uma estruturação modular, em que novas funcionalidades podem ser agregadas ou é pensada como uma

ferramenta a ser expandida que acrescenta novos horizontes para a pesquisa da área de testes em agentes autônomos.

Por fim, tem-se que os trabalhos aqui apresentados trazem grandes contribuições para o teste de agentes autônomos, seja por inferir sobre a viabilidade ou pela apresentação de uma ferramenta de teste abordando uma vertente dos programas agentes. Este trabalho tomará como base a apresentação da idealização do agente de monitoramento *ProMon* apresentado por Silveira (2013), a fim de compor a abordagem proposta com mais uma ferramenta de auxílio ao teste de agentes racionais, tentando ao máximo contornar o problema da generalidade da aplicação da proposta de solução.

Na busca por uma ferramenta de maior poder de teste, essa proposta englobará os pontos de sucesso de cada abordagem apresentada trazendo para dentro desta o modelo evolutivo de geração de casos de teste e avaliação do agente testado, bem como as preocupações acerca da viabilidade de teste e o teste direcionado as menores unidades que compõem um sistema multiagente para facilitar uma possível correção futura, tomando a plataforma de desenvolvimento JADE como auxílio para a apropriação dos padrões FIPA (*Foundation For Intelligent, Physical Agents),* uma organização de padrões da IEEE *Computer Society* que promove a tecnologia baseada em agentes e a interoperabilidade de seus padrões com outras tecnologias.

A solução aqui apresentada aborda o problema da precisão na identificação de pontos falhos que necessitem passar por correção/melhorias, de forma a trazer para o projetista informações tratadas de maior utilidade para os objetivos da fase de teste, auxiliando na garantia da robustez e do funcionamento adequado do agente.

Ainda como diferencial, a proposta de solução também se preocupa em dar maior flexibilidade ao projetista, para que este possa arbitrar sobre as características de sua solução, sendo assim, não é formulado um conjunto de regras extensas e/ou rígidas em demasia para a orientação da construção de uma solução agente que possa ser testada, mas sim apontadas pequenas pontualidades a serem usadas como padrão que serão apresentadas ao decorrer dessa proposta, bem como apresentará uma estrutura modular para que novas funcionalidades possam ser facilmente agregadas, ganhando assim notoriedade em trabalhos de expansão futuros.

4. Abordagem Proposta

Visando contribuir, de forma efetiva, para o teste de agentes racionais, este capítulo apresenta a modelagem de um agente monitor de teste baseando-se na ideia apresentada por Silveira (2013), que usa agentes racionais para o teste de agentes racionais, a fim de trazer para o projetista uma maior comodidade, agilidade e qualidade no teste de agentes autônomos. Aqui será apresentada a modelagem de um agente, *ProMon*, para monitorar e apresentar as falhas do agente testado, buscando contornar o problema da generalização de aplicação da metodologia de teste, expandindo a abrangência aos diferentes tipos de programas agentes racionais.

Para a concepção do agente de monitoramento (*ProMon*) será utilizado o framework JADE (*Java Agent Development Framework*, ver seção 4.2.1) com linguagem de programação Java, para evitar dificuldades com a operação em diferentes plataformas. A interação com o projetista será realizada por meio de arquivos de configuração e/ou interface de entrada de dados, com o intuito de facilitar a usabilidade, para que usuários leigos possam fazer uso com pouca preparação prévia.

Este capítulo iniciará apresentando uma definição geral da proposta, juntamente com conceitos básicos considerados, sendo feita uma breve introdução às tecnologias de suporte ainda não citadas neste trabalho. As características específicas de *ProMon* juntamente com a modelagem de cada módulo são apresentadas, de forma sistemática, sendo aferidas considerações sobre o desempenho de *ProMon*, pois as garantias de qualidade do teste estarão diretamente ligadas ao desempenho de *ProMon*.

4.1. Visão Geral da Abordagem

A abordagem de testes de agentes, proposta neste trabalho é baseada na descrição de um agente de monitoramento apresentada por Silveira (2013) e visa a apresentação da modelagem para este agente de monitoramento de testes. Para o funcionamento adequado da proposta de teste de agentes, considera-se a existência de quatro programas agentes: (i) agente testador (*Thestes*) concebido por Silveira (2013), (ii) agente monitorador (*ProMon*), (iii) agente testado e (iv) ambiente de tarefa:

- ***Thestes*:** Agente de resolução de problemas de seleção de casos de teste, concebido em uma estrutura de programa agente orientado por utilidade (ver seção 3.4).
- ***ProMon*:** Agente de monitoramento e diagnóstico das falhas do agente testado, que será explorado e expandido nesta proposta.
- **Agente testado:** O programa agente racional concebido pelo projetista a ser testado para inferir sobre as garantias de correto funcionamento e confiabilidade.
- **Ambiente de tarefas:** O programa ambiente de tarefa, no qual o agente testado poderá atuar durante a etapa de teste.

O agente *ProMon* proposto neste trabalho atuará, inicialmente, sobre o teste de agentes de dois tipos de programa agente: agente reativo simples e agente reativo baseado em modelo. Esses dois tipos de programa agentes são compostos pelos seguintes subsistemas de processamento de informações: ver, próximo (disponível na estrutura interna do agente baseado em modelo) e ação. Utilizamos esses dois tipos de programas agente para conceber a estrutura de *ProMon*, mas destacamos que a abordagem é expansível para os demais tipos de programa de agente.

O subsistema de percepção, *ver*, mapeia uma informação sobre a percepção do agente para um conjunto representativo abstrato útil, já que a percepção é um estado do ambiente em um dado instante tomado de forma representativa, cuja representação dependerá diretamente da abordagem empregada para a busca de soluções pelo agente, sendo a função ver a representação e obtenção das percepções a partir dos sensores do agente. O subsistema de atualização do estado interno, *próximo*, utiliza o mapeamento da percepção atual do agente e faz a atualização do estado interno mantido pelo agente, ou seja, a função *próximo* usa as informações da percepção e do estado interno atual para a construção do próximo estado. E, por fim, o subsistema de tomada de decisão, *ação*, mapeia uma informação do estado interno em uma ação correspondente, satisfazendo a ação tomada pelo agente dado seu mapeamento de estado interno.

Contudo, pode haver variações no uso do estado interno para a tomada de decisão relacionada a uma determinada ação por parte de um agente. No que diz respeito à proposta, se pode tomar como exemplo a função ação de um agente reativo simples, que usa apenas o mapeamento do estado

atual, independentemente, de estados anteriores, baseando-se apenas na percepção atual para seguir em tomada de decisão. Em contrapartida, o uso da função ação em um agente reativo baseado em modelo, que faz uso de um mapeamento do histórico de percepções do agente como estado interno, de modo a prevenir uma possível interferência das ações realizadas anteriormente na escolha da ação atual. No entanto, assume-se nesta proposta a necessidade de parâmetros de configuração, posteriormente apresentados, para a correta adaptação do funcionamento da estrutura do agente a ser testado. Com a entrada de parâmetros de configuração espera-se garantir a adequação do agente monitor para a problemática a ser resolvida pelo agente testado.

A inserção do uso da configuração por meio de parâmetros de entrada ocorre, também, pela necessidade de adequar *ProMon* para os demais tipos de programas agentes, propiciando uma expansão da proposta. A organização estrutural de *ProMon* seguirá um esquema modular, para que a introdução de novas funcionalidades seja feita de forma mais simples e rápida, por inserção de um novo módulo. Para estender o potencial de *ProMon*, este é modelado para uso não apenas com os algoritmos aqui apresentados, mas para ser facilmente estendido ao uso de outros algoritmos, sendo apresentado como uma proposta um conjunto de algoritmos de busca de estados ideais para o agente testado, significando que o alcance das soluções que são inicialmente contempladas pela possibilidade de uso desta proposta de trabalho pode ser expandida para um maior número de áreas do conhecimento e níveis mais elevados de complexidade.

4.2. Uso de Tecnologias Auxiliares

Visando maximizar o desempenho de *ProMon* e facilitar o seu desenvolvimento, um conjunto de tecnologias auxiliares é indicada como mecanismo de auxílio para a implementação. Tais indicações terão importância direta na expansão desta proposta, pois almejam a facilidade da inserção de novos módulos e funcionalidades ou o perfeito funcionamento dos módulos propostos.

A apresentação prévia dos meios utilizados para compor os mecanismos de auxílio na concepção de *ProMon* serão, inicialmente, introduzidos de forma breve, dando prioridade para as características que lhes foram decisivas para serem escalados como candidatos ao uso nesta proposta. Neste capítulo também será sugerido um ambiente de desenvolvimento que possua como característica a fácil interação com os

frameworks de auxílio a criação de *ProMon* preconizada. Para *ProMon* será importante frisar a exploração do máximo de tecnologias facilitadores atuais, pois é esperado que quando no período de uso sua manutenção e atualização sejam de fácil execução, uma vez que um novo mecanismo, algoritmo ou outra estrutura qualquer que possa contribuir seja identificada e haja o desejo ou necessidade de introduzi-lá a *ProMon*.

A organização que ocorrerá a apresentação das características dos mecanismos de auxílio, apontadas como facilitadoras da codificação de *ProMon*, seguirá um limiar didático que servirá como embasamento para a justificativa de uso nesta proposta. Logo em seguida, as ferramentas serão exploradas e inseridas no contexto do agente monitor.

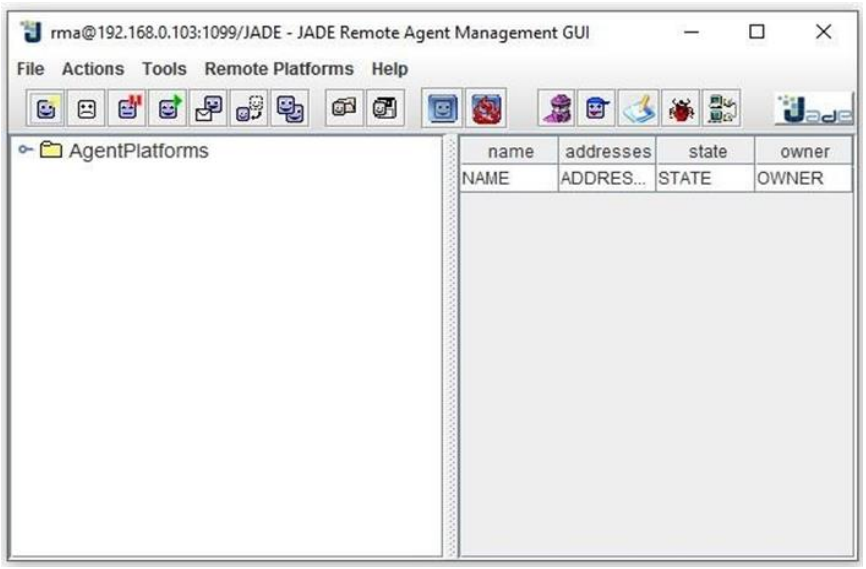
4.2.1. Framework JADE

A plataforma de código aberto JADE (*Java DEvelopment Framework*) é um software que por meio de *middleware* visa simplificar a implementação de sistemas multiagentes. Por ser plataforma totalmente desenvolvida em linguagem Java, um sistema baseado em JADE pode operar de modo distribuído e sem dependências de compatibilidade de sistemas operações (BELLIFEMINE; CAIRE; GREENWOOD, 2007).

O *middleware* JADE atende aos padrões de especificação FIPA (*Foundation For Intelligent, Physical Agents*), e cria múltiplos contêineres para os agentes, que podem estar distribuídos em rede, e podem ser executados em múltiplos sistemas. JADE também dispõe de API (*Application Programming Interface*) para gerenciar e explorar os serviços, dispondo de bibliotecas de classe para criar os agentes, por meio de herança.

Plataformas JADE, obrigatoriamente, devem conter um *Main Container*, responsável por prover os recursos da plataforma, gerenciar agentes e contêineres filhos, sendo a composição destes: o DF (*Directory Facilitator*) que é responsável por fornecer um diretório para anúncio da disponibilidade de agentes, uma espécie de serviço de páginas amarelas para a requisição de serviços e criação de confederações; o AMS (*Agent Management System*) que controla a plataforma e gerencia o ciclo de vida dos agentes; e o ACC (*Agent Communication Channel*), que gerencia a comunicação interagentes, dentro ou fora da plataforma (BELLIFEMINE; CAIRE; GREENWOOD, 2007).

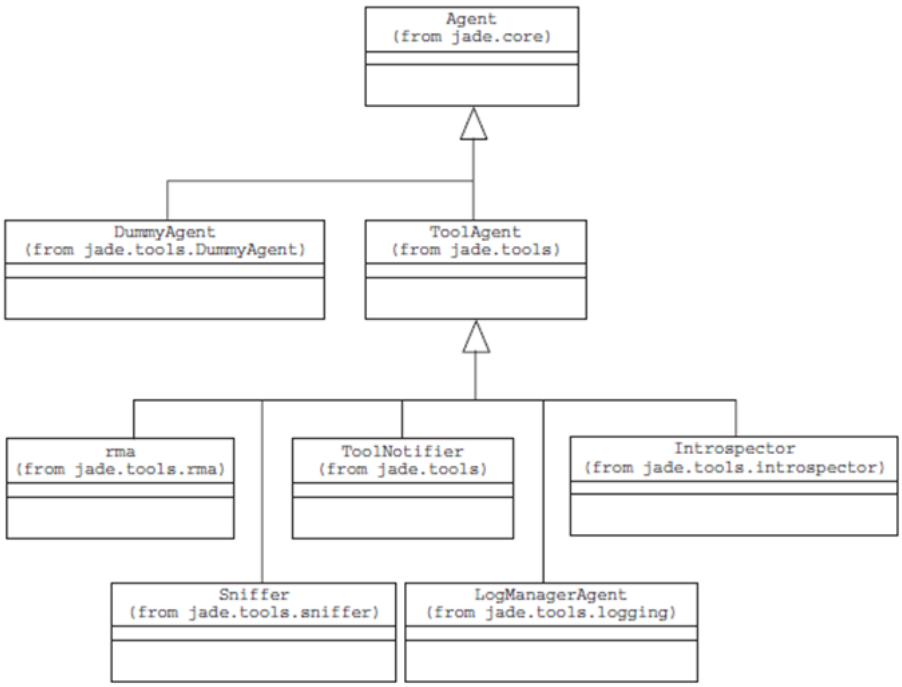
Figura 16: GUI da JADE RMA.



Fonte: Próprio autor (2017)

O JADE RMA é uma ferramenta disponível para sistema que implementam um console de gerenciamento de plataforma gráfica. Ela fornece uma interface visual para monitorar e administrar uma plataforma JADE distribuída, composta por um ou vários hosts e nós de contêiner (Figura 16). Vários RMAs podem ser lançados na mesma plataforma se for atribuído um nome de agente diferente para cada instância.

Figura 17: Diagrama de classes das ferramentas JADE.



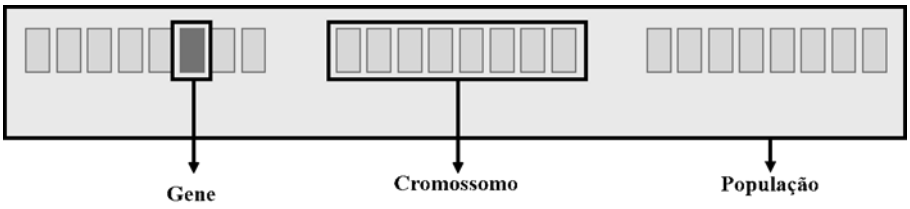
***Fonte**:* BELLIFEMINE; CAIRE; GREENWOOD, (2007).

A forma de comunicação básica entre agentes são mensagens assíncronas, contendo estrutura baseada na linguagem ACL (*Agent Communication Language*) do padrão FIPA e no paradigma *peer-to-peer*. Também há integração com diversas tecnologias web. E para identificação do agente, no meio distribuído, se tem como padrão uma nomenclatura global única. Uma visão geral das ferramentas JADE é mostrada no diagrama de Figura 17.

4.2.2. Algoritmo Genético (GA)

O Algoritmo Genético (GA) é uma metaheurística, para resolução de problemas, concebido com inspiração no processo de seleção natural das espécies. O GA faz uso de uma representação genética, baseada no conceito de codificação genética cromossômica das propriedades da solução, de forma a apresentar um conjunto de soluções possíveis como uma representação abstrata de um conjunto de DNA (*DeoxyriboNucleic Acid*), ou filamento cromossômico, mais precisamente, a representação de uma população de indivíduos, que possuem codificação cromossômica por um determinado número de genes, simulando o conjunto solução, como mostrado na Figura 18. A escolha dos indivíduos no GA se orienta por uma função de aptidão para avaliar a proximidade dos indivíduos gerados com o objetivo de solução da problemática a que é empregado (MITCHELL,1995).

Figura 18: Modelo de representação do GA.



Fonte: Próprio autor, (2017).

A forma de menor grau de complexidade de GA envolve três tipos de operadores (MITCHELL,1995). O primeiro é o de seleção que é responsável por eleger os cromossomos na população para a fase de reprodução. Quanto mais adaptado for o cromossomo, maior é a probabilidade de este ser selecionado para se reproduzir. O segundo é o operador de cruzamento, que permuta subsequências de dois cromossomos para criar dois descendentes

de características mistas. O terceiro é o de mutação que randomiza a troca de alguns genes do cromossomo.

Para encontrar a população que melhor atende ao objetivo da problemática a que é aplicado, o GA pode manter a cada iteração os indivíduos mais bem adaptados. A mutação possui papel fundamental para a garantia da diversidade da população, pois a inserção aleatória de um gene pode garantir que uma solução, cuja formação não seja possível somente a partir da população inicial, seja encontrada.

Os algoritmos genéticos têm sido usados para projetar estruturas computacionais, como autômatos e redes de classificação (MITCHELL,1995). O uso do GA, nesta abordagem, dará na seleção de episódios de comportamentos ideias por *ProMon* para comparação com os episódios de execução do agente testado em seu ambiente de atuação, na tentativa de solucionar uma dada problemática.

4.2.3. Algoritmo de Otimização Multiobjetivo: NSGA-II

Na área que abrange os softwares dotados de maior robustez, um problema cujas soluções facilmente são buscadas por meio do emprego da Inteligência Artificial (IA) é o da otimização multiobjetiva que envolve a minimização ou maximização de múltiplas funções objetivas, simultaneamente, que estão sujeitas a um conjunto de restrições, pois conta com o retorno de um conjunto de soluções e não apenas uma única saída esperada. A abordagem de resolução de problemas, por meio de agentes racionais é uma forte candidata a lidar com problemas multiobjetivos, já que se pode apresentar, por meio de algoritmos como o NSGA-II, um conjunto ótimo de soluções, mesmo em problemas cujos objetivos são conflitantes.

Problemáticas que buscam a solução para múltiplos critérios são de ordem mais complexa, quando estes critérios são conflitantes, ou seja, a maximização de um objetivo implica na minimização do outro, o que faz com que o software traga as possíveis soluções e deixe a tomada de decisão pelas partes interessadas, restrita a um conjunto seleto de menor complexidade que melhor atenda às exigências do problema. Em outras palavras, consegue-se com agentes racionais e algoritmos multiobjetivos fazer a descoberta da frente de soluções ideias de Pareto, citadas anteriormente neste trabalho.

O NSGA (*Nondominated Sorting Genetic Algorithms*) é um algoritmo de otimização de objetivo múltiplo e é uma instância de um algoritmo evolutivo do campo que compreende a computação evolutiva, sendo proposto por

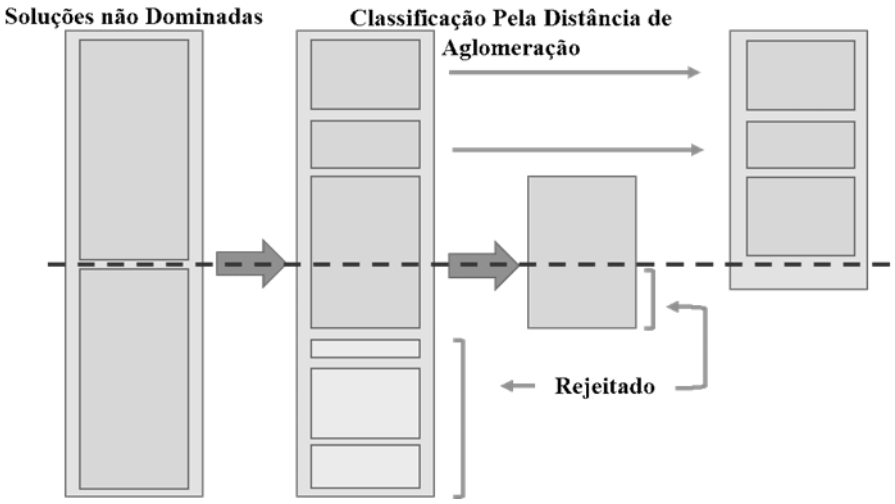
Srinivas e Deb (1994). O objetivo do algoritmo NSGA é melhorar o ajuste adaptativo de uma população de soluções candidatas a uma frente de Pareto limitada por um conjunto de funções objetivas. O algoritmo usa um processo evolutivo (HOLLAND, 1992) com substitutos para os operadores de controle evolutivo, incluindo seleção, cruzamento genético e mutação genética. A população é classificada em uma hierarquia de subpopulações com base na ordenação da dominância de Pareto. A similaridade entre os membros de cada subgrupo é avaliada na própria frente de Pareto, e os grupos resultantes e as medidas de similaridade são usadas para promover uma frente diversa de soluções não dominadas.

Proposto por Deb et al. (2000), o NSGA-II é uma modificação acrescida de enriquecimentos do NSGA, que conta com um método rápido de ordenação baseado em não-dominância que tem sido empregado para reduzir o tempo de computação para $O(MN^2)$ (em que N é o tamanho da população e M é o número de objetivos). É empregada uma abordagem que privilegia a "elite" das soluções, em detrimento das demais contidas no conjunto (elitista) e um método eficiente para determinar a distância de aglomeração do conjunto de soluções de Pareto para obter uma estimativa da densidade de soluções, envolvendo uma solução específica na população. O operador de comparação de aglomeração orienta o processo de seleção em diversas etapas do algoritmo para uma boa disseminação das soluções nas frentes ótimas. A funcionalidade ocorre na aplicação entre duas soluções com diferentes graus de não-dominância, devendo ser preferível a solução com melhor classificação, caso contrário, se ambas as soluções pertencem a mesma frente classificatória, então, deve-se preferir a solução que está localizada em uma região com menor aglomeração de soluções para manter a diversidade entre elas (conforme Figura 19). O NSGA-II aplica o elitismo para propiciar a combinação entre os melhores pais e os melhores filhos obtidos (DEB et al., 2002).

O algoritmo NSGA-II, como instância de um algoritmo evolutivo, possui em sua execução os passos fundamentais que orientam o princípio da seleção natural das espécies, começando pela formação de uma população inicial, tomando como base a função objetivo. Logo em seguida, uma avaliação é feita tomando como base o fitness atribuído a cada indivíduo gerado. Caso os critérios de satisfação não sejam atendidos com uma população, o NSGA-II aplica as métricas de aglomeração e inicia um processo de seleção, cruzamento e mutação de indivíduos para a geração de uma nova população a ser avaliada. Este ciclo é repetido até que os critérios de satisfação sejam atendidos ou que o número de gerações ultrapasse as estipuladas, como

segurança para detecção de uma estabilidade no fitness de adequação dos indivíduos das populações.

Figura 19: Modelo de representação do NSGA-II.



Fonte: Adaptada de Deb et al. (2002).

Nesta proposta, o NSGA-II desempenhará o papel de seletor de episódios ideias, em situações, cujos problemas forem multiobjetivos, a fim de munir o agente *ProMon* do conjunto de soluções que melhor atendam a demanda da problemática a que o agente testado está inserindo. Desta forma, pode-se garantir, que mesmo em condições multiobjetivas, *ProMon* será capaz de averiguar a adequação do comportamento do agente testado.

4.2.4. Framework jMetal

O projeto de concepção do framework jMetal teve seu início em 2006, como resultado da necessidade dos pesquisadores responsáveis por sua criação de terem um modo que apresentasse facilidade ao fazer uso do conjunto de metaheurísticas de otimização multiobjetivo disponíveis na literatura, bem como fosse flexível aos anseios do projetista e apresentasse extensibilidade e portabilidade para melhor atendê-lo (DURILLO et al., 2006).

O jMetal é um framework arquitetado sobre o paradigma de orientação a objetos, implementado em linguagem de programação Java para prover o desenvolvimento, experimentação e estudo de metaheurísticas, fornecendo soluções para problemáticas de otimização multiobjetivo. Outras características do jMetal incluem a geração de informações dos resultados obtidos, por meio de estatísticas geradas de modo automático e o

aproveitamento do estágio atual da computação, que propicia a disponibilidade de processadores multi-core para abreviar o tempo de execução dos ensaios, por meio de execuções paralelas (DURILLO; NEBRO, 2011).

A concepção do jMetal sobre uma arquitetura orientada a objetos acabou por ter a consequência de facilitar a abrangência de novos componentes e a reutilização dos já existentes na ferramenta. Esta concepção fornece ao usuário suporte ao uso de algoritmos, a um conjunto de problemas, a codificações adaptáveis e ferramentas indicadoras de qualidade. Ainda promove o apoio para estudos experimentais, e dispõe de um conjunto de classes de base, cuja finalidade pode variar entre a projeção de novos algoritmos, implementação de problemas ou operadores complexos, entre outros.

Neste trabalho, o framework jMetal assumirá o papel de facilitador, auxiliando na implementação dos algoritmos usados para promover as soluções necessárias para o funcionamento de *ProMon*. Espera-se que o framework contribua diretamente para a capacidade de expansão da proposta, bem como possibilite a disponibilidade de um maior número de soluções algorítmicas a serem utilizadas.

4.2.5. Integração de Tecnologias

ProMon é proposto como uma ferramenta de auxílio ao teste de agentes racionais, composto por tecnologias, como a plataforma de desenvolvimento JADE e do framework jMetal, do qual se originarão o suporte para criação do agente responsável pelo teste de subsistemas do agente testado e para busca de estados ideais, além da modelagem do conjunto de algoritmos utilizados para a realização das tarefas.

O uso dos suportes na criação de *ProMon* é motivada, principalmente, pelo cenário evolutivo da computação, em que a necessidade de atualização futura da ferramenta possa encontrar os meios mais simples de fazê-la. Já que com organização arquitetural é possível prover novas soluções, que causem alteração desde o alicerce até a realização final de sua tarefa.

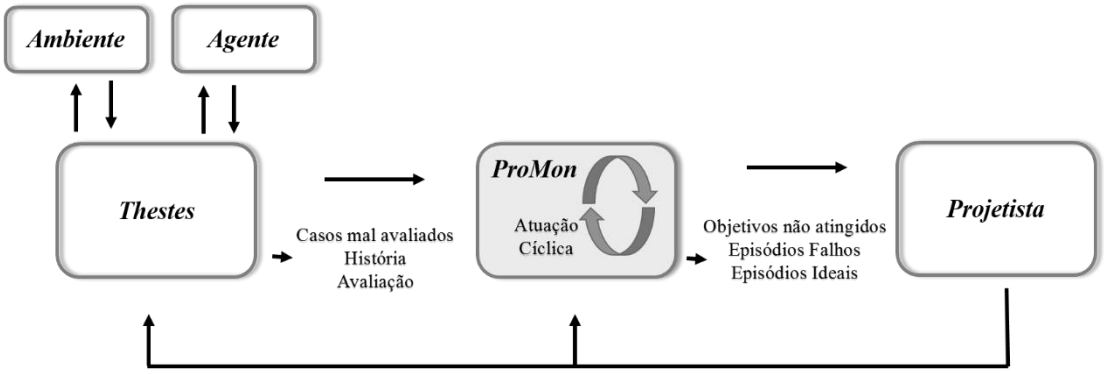
4.3. Agente *ProMon*

O agente *ProMon*, inicialmente proposto por Silveira (2013), deve ser capaz de identificar as falhas que possam ocorrer em qualquer um dos três subsistemas que compõem a estrutura interna do agente testado. Para isso, *ProMon* recebe de Thestes: (1) um conjunto de casos de teste que o agente

testado teve um comportamento inadequado, (2) as histórias do agente testado em seu ambiente de tarefa e os valores de avaliação de desempenho associados, episódio por episódio. Considerando estas informações, o agente *ProMon* deve enviar para o projetista: (1) os episódios em todas as histórias em que o agente testado falhou e os episódios ideais correspondentes, e (2) uma identificação do tipo de falha, indicando o subsistema de processamento de informação do agente testado, que ele presume esteja causando a falha, ou seja: (a) subsistema de percepção ver, (b) subsistema de atualização de estado interno próximo, ou (c) subsistema de tomada de decisão ação.

O agente *ProMon* foi planejado como um agente reativo baseado em modelo. Mais especificamente, o processo de monitoramento e diagnóstico de falha realizado pelo agente *ProMon* é proposto para ser realizado em duas etapas correspondentes ao processamento de um subsistema do tipo próximo e outro do tipo ação. Na primeira etapa, a função próximo do agente recebe as histórias associadas aos casos de teste e identifica todos os episódios contendo falhas nessas histórias. Nesse cenário, uma visão expandida da interação de *ProMon* com Thestes para o teste de subsistemas, posterior a seleção de casos de teste é apresentada na Figura 20.

Figura 20: Visão da iteração de ProMon.



Fonte: Adaptado de Silveira (2013).

Para prover um padrão de comparação a ser empregado sobre os episódios, *Ei*, do agente testado (denotaremos de *AgenteEx*) o agente *ProMon* utiliza uma versão onipresente do *AgenteEx* (totalmente observável) denotado por *AgenteEx**, concebida tomando como base para a função ação uma coleção de algoritmos, providos pelo auxílio do jMetal, que apresentam em sua definição a condição de garantia de encontrar soluções quando estas são possíveis, gerando assim um conjunto *Ei** de episódios ideias, contendo nesse conjunto de episódios o subconjunto de estados dos subsistemas a serem

avaliados. Com isso, se pode ter uma comparação entre os *Ei** episódios ideais gerados por *AgenteEx** e os episódios *Ei* gerados por *AgenteEx*.

Sendo contidos nos episódios *Ei* e *Ei*,* os pares (percepção, ação) que são necessários para o conhecimento do estado dos subsistemas do *AgenteEx* e do *AgenteEx**, respectivamente, possibilitando comparar cada um dos subsistemas, avaliando-os de forma individual, uma vez que se pode concluir que os episódios ideias *Ei** propiciam também o conhecimento do conjunto de estados ideias dos subsistemas ver: *Vi*, próximo: *Pi* e ação: *Ai,* em cada instante da atuação, sendo esses estados ideais denotados respectivamente por *Vi*, Pi*, Ai*,* oferecendo assim uma forma segura de avaliação do *AgenteEx* e de seus subsistemas.

Ao fim do teste, que transcorre durante a atuação do *AgenteEx* em seu ambiente, o agente *ProMon* reúne as informações advindas das comparações resultantes entre *Ei* e *Ei** para confeccionar um relatório apontando as falhas do *AgenteEx* ao longo de sua execução, bem como o diagnóstico contendo a indicação dos subsistemas responsáveis pelas falhas. Dessa forma, indicando ao projetista o local em que ele deve ter maior atenção ou fazer possíveis alterações.

Seguindo mais precisamente a formulação de Silveira (2013) para o teste de agentes reativos, o conjunto de episódios ideais em uma iteração *K*, $Ei^{K}*$, é formado por todos os episódios possíveis de serem produzidos por *AgenteEx** na iteração, $Ei((Vi^{K}*, Ai^{K}*))$, que satisfazem pelo menos uma das duas condições abaixo:

(1) Para todo atributo *m* na medida de avaliação de desempenho:

$av_m (Ei((Vi^{K}*, Ai^{K}*))) \geq av_m (Ei((Vi^{K}, Ai^{K})))$;

(2) $Ai^{K}*$ é melhor que ou é equivalente a Ai^{k} considerando o ponto de vista do projetista.

E, consequentemente, episódios $Ei((Vi^{K}, Ai^{K}))$, produzidos pelo agente testado *AgenteEx* que não pertencem ao conjunto $Ei^{K}*$, devem compor o conjunto de episódios com falha Ei_{falhas}. Sendo assim, pode-se concluir que somente há a necessidade de conhecimento dos estados ideais, por parte do *AgenteEx** concebido por *ProMon*, que desta forma pode realizar uma busca finita de soluções, usando o princípio indutivo de negação para afirmar que se um episódio não é encontrado como ideal, é porque pertence aos episódios falhos.

Para que a proposta seja efetivada é necessário voltar maior atenção para sua aplicabilidade, pois sua aplicação depende do correto conhecimento do conjunto de estados ideais, sendo que estes estados ideais devem ofertar

os melhores graus de satisfação do desempenho de um agente testado na resolução de problemas. Porém, na ausência de uma única resposta ideal esperada, *ProMon* pode avaliar com base na melhor solução possível para o problema.

ProMon será escrito em linguagem de programação Java, dando maior portabilidade entre sistemas operacionais, além de propiciar o suporte pela plataforma *JADE* e auxílio do framework jMetal. *ProMon* necessitará conter uma coletânea de algoritmos capazes de devolver os episódios ideais para comparação com os episódios do agente testado. Inicialmente é planejada a inserção de dois algoritmos base para os testes preliminares, sendo utilizados o GA e o NSGA-II. Os algoritmos deverão ser aplicados mediante configuração prévia, baseada em entradas dadas pelo projetista.

4.3.1. Módulo de Configuração

Como a proposta de *ProMon* visa atender ao teste de agentes, cuja resolução de problemas possa ocorrer em um escopo de abrangência genérica, com atuação que atenda os mais rigorosos anseios de eficiência e eficácia, nota-se que a apresentação prévia de parâmetros de entrada para a correta adaptação do comportamento de *ProMon* se faz necessário. Esses parâmetros devem considerar aspectos chave para a formação de um modelo de agente a ser posteriormente implementado de forma automática por *ProMon* para a função de *AgentEx** na busca pelos episódios ideais *Ei**. Esses parâmetros de configuração atendem a necessidade de conhecimentos específicos que são:

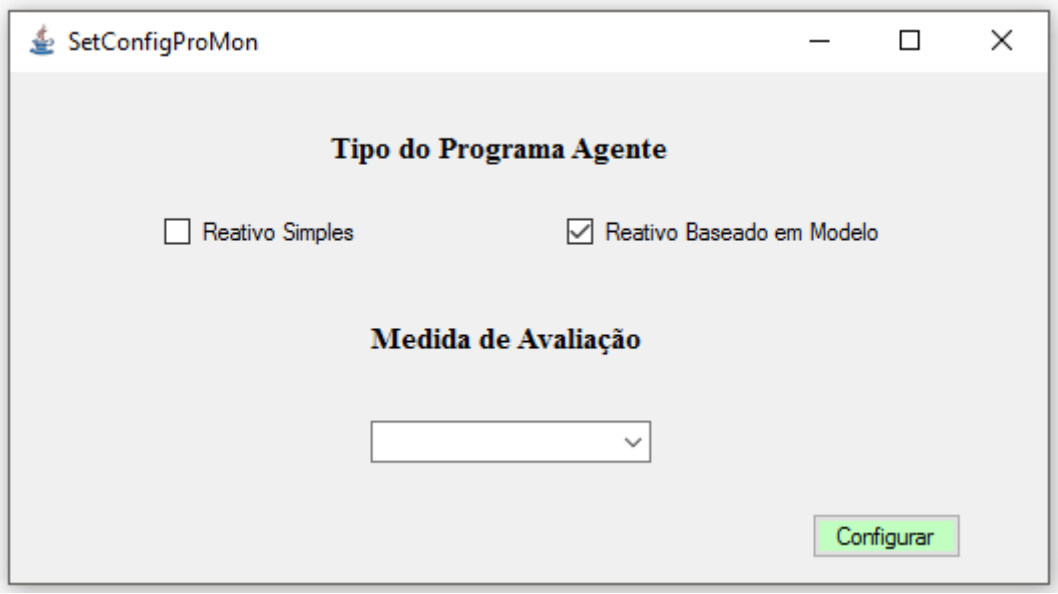
- **Tipo de programa agente:** Faz referência ao tipo de programa agente utilizado, a princípio se consideram os tipos: reativo simples e baseado em modelo.
- **Agente:** um programa agente racional que está passando pela fase de teste.
- **Medida de avaliação:** critério objetivo que é usado pelo agente para tomar as decisões de maior sucesso, quando não for possível realizar todos os objetivos. A medida de avaliação considera o resultado que é desejado da atuação no ambiente.
- **Casos de teste:** são um conjunto de condições usadas para teste de software. Podem ser elaborados para identificar defeitos na estrutura interna do software, ou ainda, do funcionamento destes componentes de maneira integrada por meio de situações que exercitem adequadamente

todas as estruturas utilizadas na codificação.

- **Histórias:** conjunto de interações do agente com seu ambiente de tarefa.
- **Avaliação:** campo cuja função é dizer quanto cada um dos objetivos implícitos na medida de avaliação de desempenho foi satisfeito em cada interação e a utilidade final da história do agente no ambiente.

É importante frisar que o padrão de mensagens adotado para o fluxo de comunicação é o *ACLMessage*, para que haja concordância entre o fluxo de comunicação esperado por essa proposta e o utilizado pelo projetista de um agente a ser testado. As configurações de entrada que não vierem por meio da interação entre *ProMon* e *Thestes* devem ser aplicadas diretamente no arquivo config*ProMon*.xml, por intermédio de uma função SetConfig*ProMon*, que fornecerá um meio simplificado e intuitivo de apresentação dos parâmetros de entrada, como é mostrado na proposta de GUI da Figura 21.

Figura 21: Visão da interface de SetConfigProMon proposta.



Fonte: Proprio autor, (2017).

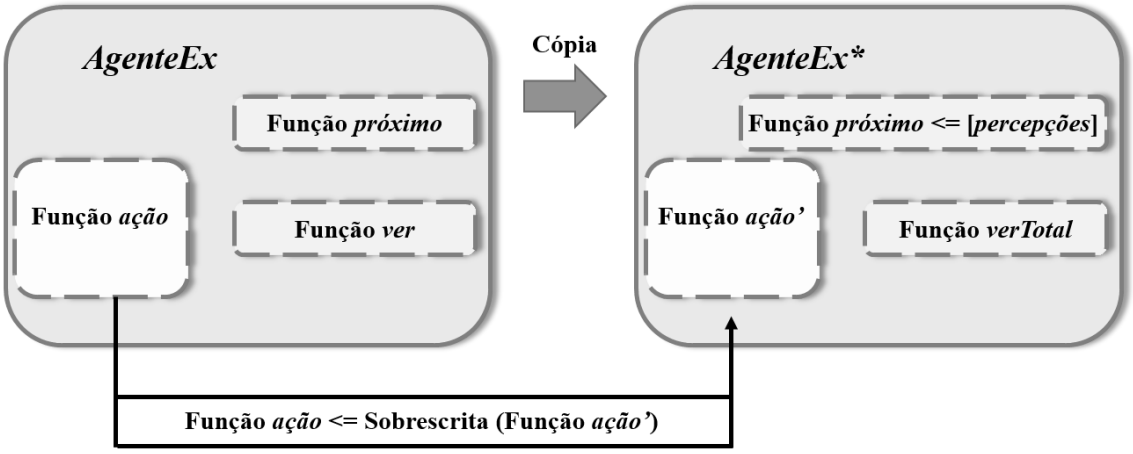
A escolha da onipresença do agente, que busca retornar todos os episódios ideais, ocorre para que a falta de informação sobre o ambiente não possa atrapalhar a atuação do *AgenteEx**. Além disso, é preciso retratar que a versão onipresente não é uma cópia exata da implementação do agente a ser testado, mas sim de seus objetivos, pois sua atuação dependerá da sobrescrita de sua função *ação* por *ProMon*. A forma de busca pelos episódios

ideias deve ser tida de maneira que se possa minimizar o tempo e recursos computacionais gastos, contando com critérios de parada e persistência bem definidos.

Como o módulo de configuração de *ProMon* é o responsável pela montagem de uma arquitetura que, por sua vez, é a base de formação do *AgenteEx**. Para que esta arquitetura seja confeccionada com sucesso é preciso que haja a formação de uma cópia do *AgenteEx,* melhorada por *ProMon* que cuidará da modificação da visibilidade do ambiente para total e da composição da função *ação*. O fato de *AgenteEx** não ser uma cópia fiel de *AgenteEx* é a garantia de que não estará exposto a mesma probabilidade de falha.

A criação da cópia que originará *AgenteEx** será orientada pelo arquivo de configurações solicitado, inicialmente, por *ProMon* acrescido das entradas fornecidas por *Thestes*, para que com todos os detalhes conhecidos sobre o agente se possa estabelecer as melhores modificações para a cópia de *AgenteEx* para a criação do *AgentEx** por *ProMon*, tomando como base seus recursos disponíveis. Os aspectos do procedimento proposto podem ser vistos na figura 22, em que é exibido o tratamento da cópia de *AgenteEx* por *ProMon* para a montagem da versão *AgenteEx**.

Figura 22: Proposta de criação da *AgenteEx**.



Fonte: Próprio autor (2017).

Os principais desafios de *ProMon* nesta fase resumem na preocupação em prover ao *AgenteEx** ferramentas de interpretação de suas percepções, bem como meios para entender as consequências de suas ações, de modo a ser capaz de aferir o grau de sucesso delas. Embora *ProMon* não execute a captura de informações necessárias diretamente do fluxo de comunicação no cenário de atuação (atribuição de *Thestes*), ele considera que

as mensagens seguem o padrão *ACLMessage* pré-estabelecido anteriormente, bem como usa sempre a máxima observabilidade de seu ambiente. O padrão de comunicação entre agente e ambiente nesta proposta deve ocorrer de forma cíclica onde o agente ambiente envie o estado do ambiente para o agente atuador e este o retorne uma ação desejada que após executada o ciclo pode ser retomado.

A escolha da regra de ação para o *AgenteEx** que deve ser realizada por *ProMon* por afinidade de atuação, por exemplo, em problemas mono objetivo é sugerido inicialmente o uso da metaheurística GA, assim como, o NSGA-II é sugerido para resolução de problemas multiobjetivo como mostrado na Figura 23, em que a escolha do algoritmo para *AgenteEx** é exemplificada em pseudocódigo com base na afinidade de atuação para compor as regras de ação. Como agentes reativos baseados em modelo além das funções *ver* e *ação* fazem uso da função *próximo*, estabelecemos que a função próximo do *AgenteEx**, quando este for baseado em modelo, deverá receber o conjunto de percepções do agente em sua atuação.

Figura 23: Proposta de seleção de regra de ação do *AgenteEx**.

```
programa
{
 função regraDeAção(entrada funçãoAvalição) retorna Algoritmo{

        se problema igual a multiobjetivo{
                        selecione NSGA-II
                        retorne NSGA-II
        }
        se problema igual a mono objetivo{
                        selecione GA
                        retorne GA
        }
} }
```

Fonte: Próprio autor (2017).

Ao final da execução do módulo de configuração uma versão do *AgenteEx* modificada, o *AgenteEx**, estará completamente funcional e pronto para a fase de busca de estados ideais do próximo módulo. É importante notar que para esta proposta considera-se que o projetista tenha informado as características e entradas necessárias de acordo com a descrição de solicitação das mesmas para a prevenção de erros ou falhas na atuação de *ProMon.* O AgenteEx* formada no módulo de configuração será a peça principal para o módulo de inteligência, do qual seguirá a atuação que

antecede o teste comparativo de *AgenteEx*.

4.3.2. Módulo de Inteligência

O modo de inteligência de *ProMon* será o responsável por uma parte essencial desta proposta: a captura de todos os estados ideias *Ei** que serão utilizados para a detecção de pontos de falha na atuação do *AgenteEx*, por meio de comparação com os estados *Ei* gerados a partir da atuação do *AgenteEx* em seu ambiente.

Os estados ideais *Ei** são resultado da iteração do *AgenteEx** com seu ambiente de atuação, e cada episódio deve trazer o par de conjuntos de percepções e ações que levem o *agenteEx** a um estado desejável. Desse modo se garante que qualquer episódio que leve a uma solução do problema seja considerado. Os episódios *Ei* resultantes de *AgenteEx* são constituídos do par percepção, ação *Ei*(Vi, Ai), porem considerando que podem haver mais de uma ação possível para uma percepção, assim como podem haver múltiplas formas de perceber o mesmo estado de ambiente os episódios ideais são compostos pelos conjuntos *EI**([*Vi*], [*Ai*]), pois se tomarmos como exemplo uma percepção *x*, podemos ter a partir desta *n* ações que podem levar a um estado satisfatório.

Figura 24: Proposta de codificação da cópia de agente a ser testado.

```
programa
{
 funcao AgenteAbstrato (percepção) retorna ação
            {
            estatico: regras, estado
            estado <= AtualizaEstado(estado, ação, percepção)
            regra <= ApropriacaoDeRegra(estado, regras)

            ação <= RegraDeAção (regra)
             ação_I <= ação
            RegraDeAção_X <= RegraDeAção – ação

            enquanto ação_J ≥ ação_I faça{
                        ação_J <= RegraDeAção_X(regra)
                        EstruturaEp <= ação_J, estado, amb
                        RegraDeAção_X <= RegraDeAção_X – ação_J
                        }
            retorne ação
            }
}
```

Fonte: Próprio autor (2017).

Para garantia da exploração de todos os possíveis episódios ideais é

incluso uma estrutura de repetição que busca por ações potencialmente desejáveis dentro do conjunto de ações que podem ser tomadas pelo *AgenteEx**, sendo feita de forma que o *AgenteEx** faça uma segunda escolha caso a primeira não fosse possível, decrementado da ação já escolhida do conjunto de ações possíveis, como pode ser visto na figura 24. Esse laço repetitivo cria uma estrutura que armazena as ações candidatas, para que após sua execução no ambiente o grau de satisfação seja averiguado pela função objetivo do agente a fim de verificar se ela não leva a um estado ideal, essa averiguação é feita com comparação ao grau de satisfação da primeira escolha, pois se essa segunda oferecer grau satisfatório pelo menos igualitário seu episódio de atuação deve ser tomado como um *Ei**.

Podemos apresentar como exemplo geral da busca de estados ideais a atuação de um GA que buscar findar sua atuação com uma população (ver seção 4.2.2) que represente resoluções para a problemática, ou seja estados ideais. Os algoritmos de embasamento genético são escolhidos por que embora possam haver muitos tipos de problema o objetivo de *ProMon* é maximizar o grau de satisfação frente a função objetivo de *AgenteEx**, o que faz de algoritmos otimização candidatos ideais para uso em nossa problemática, e sendo os algoritmos de embasamento genético voltados a problemas de otimização esses foram os escolhidos iniciais.

Ao finalizar a atuação do módulo de inteligência é formado um conjunto contendo *n* episódios ideias, lembrando os episódios são formados da dupla percepção e ação, e sendo que *n* deve ser menor que o número total de deliberações do *AgenteEx** e não vazio, pois as deliberações do *AgenteEx** são feitas de forma que encontre, pelo menos uma vez, na estrutura de busca por candidatos a ideais, um estado não satisfatório a ser descartado, caso contrário todo estado seria ideal para a resolução do problema.

Com a finalização da busca pelo conjunto de episódios ideais, a fase de atuação que antecede o teste comparativo de *AgentEx* é findada (fases pré-testes), pois após essa fase todos os pré-requisitos de teste já foram devidamente encontrados (episódios ideais) e *ProMon* se encontra pronto para deliberar acerca da atuação do *AgenteEx*. Somente após essa etapa de formação de um conjunto de episódios ideais, as regras condicionais comparativas podem ser aplicadas.

4.3.3. Módulo de Detecção de Falhas

Concluídas as fases pré-teste de *ProMon*, o agente está pronto para realizar o teste do *AgenteEx*. O responsável por esta tarefa é o módulo de detecção de falhas, em que os episódios gerados a partir da atuação do *AgenteEx* em seu ambiente serão comparados por meio de regras com os episódios contidos no conjunto de episódios ideais gerados pelo *AgenteEx**, para que ao final se identifique uma possível falha e se explore qual o

subsistema originou a falha.

Para a fase comparativa é considerado que para o desempenho satisfatório do agente *AgenteEx*, seus episódios devem pertencer ao conjunto de episódios ideais. Caso não pertença ao conjunto de episódios ideias, o episódio é classificado como falho e, posteriormente, são realizadas verificações de quais subsistemas estão causando a falha, seguindo regras condicionais de comparação propostas por Silveira (2013), como demonstrado na Figura 25.

Figura 25: Regras condição-ação de agente *ProMon* para agentes reativos.

(1) se Agent for reativo simples e a Condição (1)' for satisfeita então:
"Condição (1)' foi satisfeita."
"Falha na função ver **se**
$ver(P^K) \neq ver^*(P^K) = Estado^K$ **e** $ação(Estado^K) = ação^*(Estado^K) = A^{K*}$, ou
falha na função ação **se**
$ver(P^K) = ver^*(P^K) = Estado^K$ **e** $ação(Estado^K) \neq ação^*(Estado^K) = A^{K*}$, ou
falha nas funções ver e ação **se**
$ver(P^K) \neq ver^*(P^K) = Estado^K$ **e** $ação(Estado^K) \neq ação^*(Estado^K) = A^{K*}$."

(2) se Agent for reativo simples e a Condição (2)' for satisfeita então:
"Condição (2)' foi satisfeita."
"Falha na função ver **se**
$ver(P^K) \neq ver^*(P^K) = Estado^K$ **e** $ação(Estado^K) = ação^*(Estado^K) = A^{K*}$, ou
falha na função ação **se**
$ver(P^K) = ver^*(P^K) = Estado^K$ **e** $ação(Estado^K) \neq ação^*(Estado^K) = A^{K*}$ ou
falha nas funções ver e ação **se**
$ver(P^K) \neq ver^*(P^K) = Estado^K$ **e** $ação(Estado^K) \neq ação^*(Estado^K) = A^{K*}$."

(3) se Agente é baseado em modelos e Condição (1)' foi satisfeita então:
"Condição (1)' foi satisfeita."
"Falha na função próximo **se**
$próximo(ver(P^K)) \neq ver^*(P^K) = Estado^K$ **e** $ação(Estado^K) = ação^*(Estado^K) = A^{K*}$, ou
falha na função ação **se**
$próximo(ver(P^K)) = ver^*(P^K) = Estado^K$ **e** $ação(Estado^K) \neq ação^*(Estado^K) = A^{K*}$, ou
falha nas funções próximo e ação **se**
$próximo(ver(P^K)) \neq ver^*(P^K) = Estado^K$ **e** $ação(Estado^K) \neq ação^*(Estado^K) = A^{K*}$."

(4) se Agent é baseado em modelos e Condição (2)' foi satisfeita então:
"Condição (2)' foi satisfeita."
"Falha na função próximo **se**
$próximo(ver(P^K)) \neq ver^*(P^K) = Estado^K$ **e** $ação(Estado^K) = ação^*(Estado^K) = A^{K*}$, ou
falha na função ação **se**
$próximo(ver(P^K)) = ver^*(P^K) = Estado^K$ **e** $ação(Estado^K) \neq ação^*(Estado^K) = A^{K*}$ ou
falha nas funções próximo e ação **se**
$próximo(ver(P^K)) \neq ver^*(P^K) = Estado^K$ **e** $ação(Estado^K) \neq ação^*(Estado^K) = A^{K*}$."

Fonte: Silveira (2013).

Depois de identificado o subsistema a ser diagnosticado como causador da falha, o módulo de identificação escreve no arquivo de teste a avaliação contendo todas as informações pertinentes aos episódios do *AgenteEx* e o conjunto ideal a ser atingido, bem como as informações referentes ao subsistema que apresentou a falha.

Na fase de teste, as regras apresentadas seguem os moldes das duas condições apresentadas na seção 4.3, cuja identificação do subsistema falho é feita por comparação direta. Como as garantias do teste estarão ligadas, diretamente, com a correta geração de episódios ideais, os algoritmos evolucionários suportados pelo jMetal serão a base inicial de busca pelos mesmos, pois com uma ferramenta de suporte, se garante que a implementação seja feita com maior segurança quanto à corretude e sua organização arquitetural.

Figura 26: Exemplo de saída do relatório XML.

```
▼<relatorio_de_teste>
  ▼<configuracoes>
     <tipoAgente>REATIVO SIMPLES</tipoAgente>
     <visibilidadeAmbiente>PARCIAL</visibilidadeAmbiente>
     <etc>...</etc>
   </configuracoes>
  ▼<episodio>
     <numero>EPISODIO01</numero>
     <EP>(VER,ACAO)</EP>
     <EPideais>(ver1,acao1)(ver2,acao2)(ver3,acao3)...</EPideais>
     <avaliacao>FALHO</avaliacao>
   </episodio>
  ▼<episodio>
     <numero>EPISODIO02</numero>
     <EP>(VER,ACAO)</EP>
     <EPideais>(ver1,acao1)(ver2,acao2)(ver3,acao3)...</EPideais>
     <avaliacao>IDEAL</avaliacao>
   </episodio>
 </relatorio_de_teste>
```

Fonte: Próprio autor (2017).

Outra particularidade que o módulo de detecção de falhas tem que lidar é a possível presença de um primeiro episódio não desejável que acarreta uma sequência de episódios não esperados, apontando para uma falha múltipla dos subsistemas que compõem um agente. Para lidar com esta situação, não corriqueira, a primeira identificação de falha deve ser tratada com maior cuidado pelo projetista, pois essa pode comprometer o sucesso de

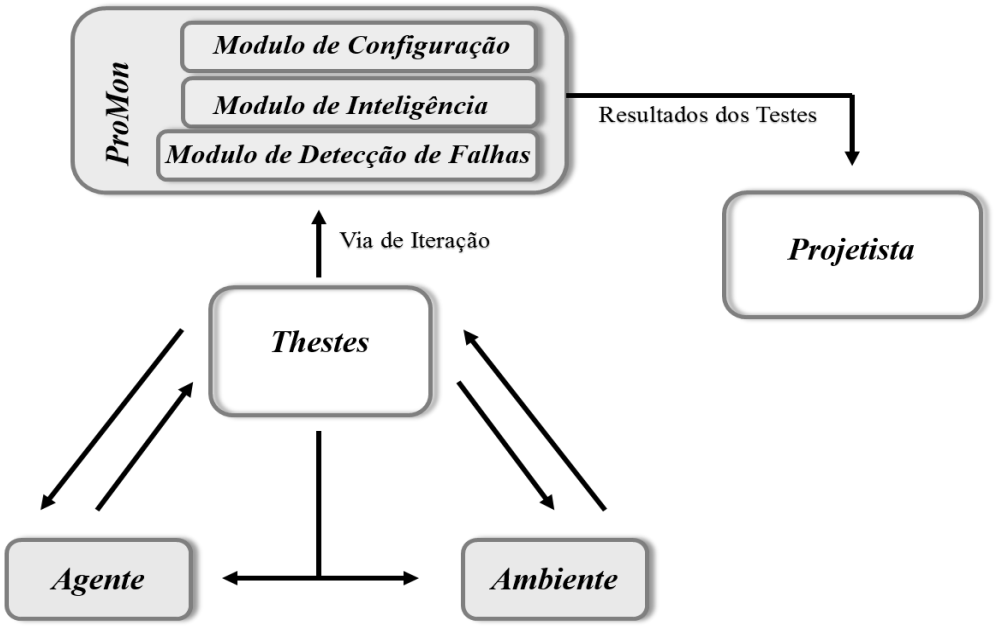
um agente em sua execução. No entanto, em nada comprometerá o teste automático de agentes, pois mesmo que induzidos por uma primeira falha os episódios não deixam de conter estados não desejáveis, e esta vertente é identificável pelo método comparativo.

Para melhor entendimento do diagnóstico de falhas do agente pelo projetista, o módulo de detecção de falhas além de exibir em tela os resultados do teste, deve apresentar um arquivo xml de relatório de teste em que todas as informações pertinentes ao teste são apresentadas, desde as configurações aplicadas aos episódios de teste, até as identificações de subsistemas falhos, trazendo mais possibilidades de análises após teste pelo projetista (Figura 26).

4.3.4. Integração de *ProMon*

A estrutura de *ProMon* é findada como uma proposta subdividida em três módulos essenciais, em que há uma cooperação e dependência de atuação entre os mesmos: (i) modulo de configuração, (ii) modulo de inteligência e (iii) módulo de detecção de falhas. *ProMon* atua em iterações com o agente *Thestes* para receber ou dar colaboração no teste de agentes, a iteração é representada na figura 27. *ProMon* auxiliará o teste de agentes, por meio da análise de episódios repassados pelo agente Thestes que já são demarcados como episódios com apresentação de falhas, onde seu papel será a confirmação da falha e a indicação de seu ponto origem específica meio aos subsistemas que compõem um agente.

Figura 27: Arquitetura de Integração *ProMon.*

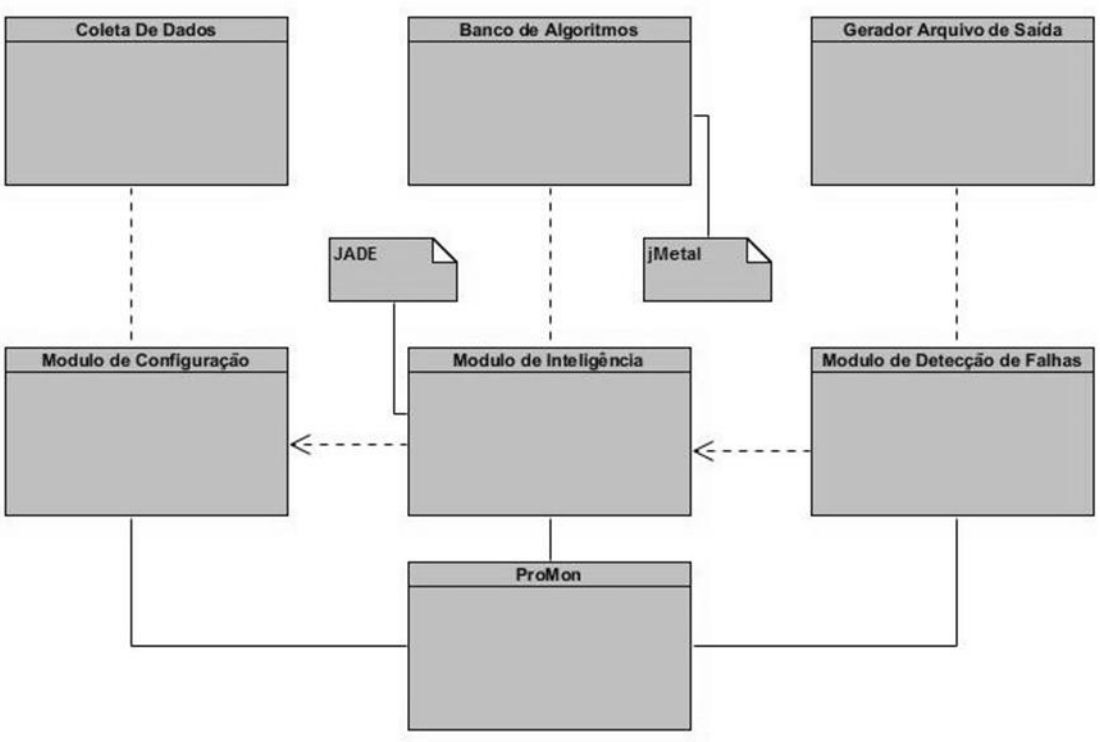


Fonte: Próprio autor (2017).

O framework jMetal e a plataforma JADE na proposta de *ProMon* são usados para enriquecer a possibilidade de uma ampliação posterior. A princípio, o uso de algoritmos evolutivos, como GA e NSGA-II são suficientes para prover um leque de testes sobre a eficiência e eficácia desta proposta. Portanto, pode haver uma expansão do conjunto de algoritmos adaptáveis disponíveis em *ProMon* para implementação do *AgenteEx**, como base de apoio à busca dos episódios *Ei** ideais.

Uma visão geral da modelagem da proposta pode ser vista no diagrama da Figura 28, que contém os blocos que compõem *ProMon* em sua devida organização arquitetural. Lembrando que existem dependências de atuação entre os módulos de composição de *ProMon,* pois o primeiro módulo a realizar a função passa para o próximo dados a serem empregados em um novo passo, como por exemplo a estrutura de configuração passada do módulo de configuração ao módulo de inteligência para a concretização do agente *AgenteEx**.

Figura 28: Diagrama de Blocos Componentes de *ProMon*.



Fonte: Próprio autor (2017).

Com todo o planejamento é almejada a meta de ao final da

implementação ter um agente capaz de promover, de forma efetiva, a identificação de subsistemas falhos de agentes aplicados aos mais diversos tipos e variações de problemáticas.

4.3.5. Modelagem de Agente Para o Teste

Durante o decorrer deste trabalho são apresentadas as preocupações com a liberdade do projetista na modelagem de suas soluções baseadas em agentes racionais. Essa preocupação resume-se em limitar o mínimo possível o formato de estrutura que o projetista tenha que seguir para que a ferramenta de teste *ProMon* seja útil. Por este motivo, esta proposta considera a criação de um agente com base nos padrões estabelecidos pela FIPA e não fecha o leque de possibilidades de aplicação em primeira instância, mas com o ressalvo de que esta primeira fase da proposta visa o teste de agentes para os tipos de programas reativos simples e baseados em modelo, mas que almeja fortemente a expansão para estruturas diferenciadas.

Por fim, se tem como restrição única o processo comunicativo entre agente e ambiente de atuação, em que o agente deve receber do ambiente o seu estado e enviar a resposta ao mesmo, contendo sua deliberação de ação. Com isso, é garantido que a liberdade do projetista seja mantida e que *ProMon* possa atuar contento todos os requisitos necessários para teste, trazendo maior diferencial e empregabilidade para a proposta.

4.3.6. Avaliação de Desempenho de *ProMon*

Para que haja uma correta avaliação do desempenho de *ProMon* é proposta sua utilização dentro de ambientes controláveis e de possibilidades finitas, em que os episódios ideias possam ser identificados e tabulados por seu projetista, a fim de se validar o funcionamento do agente monitor em sua atuação, pois uma vez que se apresenta munido dos episódios ideais, as deliberações de *ProMon* podem ser testadas uma a uma, para conferência de corretude.

O agente *ProMon*, aqui descrito, pode ser aplicado para avaliar o funcionamento em qualquer tipo de programa agente abordado nesta proposta, cuja complexidade final dependerá diretamente do algoritmo utilizado para a concepção do *AgenteEx**, tornando a complexidade de *ProMon* diretamente proporcional à complexidade do algoritmo empregado, e consumindo memória que seja suficiente para manter armazenado os *Ei**

episódios ideais, necessários para a avaliação de um dado conjunto de episódios *Ei* do *AgenteEx*.

5. Proposta Avaliativa

Para que haja uma correta avaliação de *ProMon* é necessário que se conheçam todas as possíveis deliberações ótimas, pois uma escolha não ótima por *ProMon* representaria uma falha na atuação de algum de seus módulos ou do algoritmo usado para a busca dos episódios. Sendo assim, este trabalho também apresenta uma proposta de avaliação de *ProMon*.

5.1. Cenários Propostos

Para facilitar o experimento de teste, utilizamos um ambiente de soluções finitas e tabuláveis, cuja escolha é realizada pela capacidade de inserção de novos desafios e expansão do campo de atuação. Para início dos trabalhos de teste de *ProMon* se toma como suporte o desafio do Mundo do Wumpus, que foi proposto como uma nova tipologia de jogo para computador em 1972, por Gregory Yob. Consistindo de um tipo de labirinto que é habitado por um monstro, no qual o jogador (que trataremos como um agente) tem a missão de se infiltrar e nesse tipo de caverna labirinto atua em busca de atingir metas, que são dificultadas por uma série de adversidades e pela própria presença do monstro Wumpus (YOB,1975).

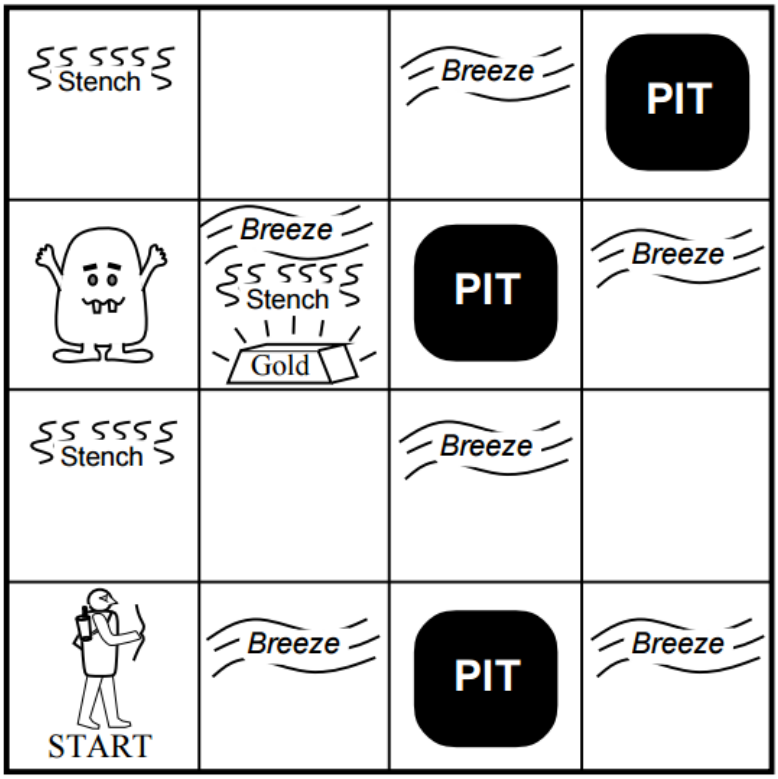
Como o desafio do Mundo do Wumpus acontece em um ambiente de dimensões finitas, o número de possíveis ações também é finito, o que limita o conjunto de episódios ideais a um número reduzido, pois dentre as ações, muitas irão compor os episódios não desejáveis. Outra motivação para a escolha deste desafio é sua vertente adaptativa, em que a descrição original do jogo pode ser alterada para facilitar a implementação do desafio e maior intimidade no acompanhamento dos testes de *ProMon*.

A primeira fase de experimentos pode conter um ambiente em que um agente jogador a ser testado possa atuar mais tranquilamente, com desafios fixos e facilmente controláveis como, por exemplo, a colocação de barreiras ao invés de morcegos, que podem transportar o jogador para outro lugar qualquer, o que conferirá maior controle sobre os estados do ambiente e sobre as respostas desejadas. Vale lembrar que no teste de *ProMon*, em que o projetista detenha todos os estados desejáveis, é possível apenas simular uma entrada de atuação de um agente a ser avaliado, o que propicia ao projetista maior domínio frente às deliberações de *ProMon* acerca da “atuação” desse

agente a ser avaliado, já que o mesmo pode simular pontos de falha específicos.

Como cenário inicial de avaliação é apresentado um cenário simples do desafio do Mundo de Wumpus em um ambiente de dimensões 4x4, com objetivo de chegar até o ouro sem cair nos poços ou ser pego pelo Wumpus, sendo que o agente só pode fazer movimentos para quadrados adjacentes ao seu e nos quadrados ao redor de poços é sentido uma brisa, e nos que ficam ao redor do monstro Wumpus um odor, o agente ainda possui uma flecha que pode atirar em um quadrado adjacente na tentativa de matar o Wumpus que se acertado emite um som ouvido pelo agente, o cenário é mostrado na Figura 29, cada ação terá uma penalidade de -1 ponto, cair no poço zera os pontos assim como ser pego pelo Wumpus, o agente terá 12 pontos iniciais para realizar seu objetivo.

Figura 29: Ambiente de tarefas ilustrativo do Mundo de Wumpus.



Fonte: Russell e Norvig (2004).

As possibilidades de falhas no ambiente de Wumpus são variadas, mesmo com uma brisa e o odor como alertas de perigo as decisões são complexas se um agente possuir apenas uma visibilidade parcial do ambiente. Já uma cópia de agente concebida por *ProMon* pode por meio da característica de observabilidade total, tomar ações melhor fundamentadas e

desviar dos perigos com maior facilidade. Se tomarmos a posição inicial do agente no ambiente de Wumpus da Figura 28 teríamos dois caminhos de menor custo para o objetivo *i** = (direita, para cima, para cima) e *ii** = (para cima, direita, para cima), o que é fácil de identificar ao ter uma visão total do ambiente o que corresponderia a solução do agente implementado por ProMon, mas ao se restringir a uma visão parcial notamos que o número de possibilidades comportamentais aumenta consideravelmente, pois para compor as duas primeira ações (número mínimo para levar ao final do jogo, neste caso por derrota) temos oito possibilidades: *i* = (para cima, flecha); *ii* = (para cima, para cima); *iii* = (para cima, direita); *iv* =(direita, direita); *v* = (direita, para cima); *vi* = (direita, flecha); *vii* =(flecha, para cima); *viii* (flecha, direita).

Vistos os exemplos acima teríamos que um agente que teria oito possibilidades de escolha e apenas duas corretas o que representa 75% de chance de falha na escolha da ação, enquanto o agente melhorado criado por *ProMon* teria como encontrar com menor esforço os dois caminhos de menor custo que deveriam ser tomados pelo agente a ser testado. Tomando o caminho ideal como parâmetro *ProMon* é capaz de extrair de sua história os estados dos subsistemas *ver*, *próximo* e *ação* que pelas métricas comparativas do módulo de detecção de falhas indicam o ponto de falha do agente testado, comprovando a viabilidade do funcionamento de *ProMon*.

Ao decorrer da fase de testes e da adequação de *ProMon*, o desafio pode ser encorpado, apresentando novas provocações ao desempenho de um agente, como por exemplo a adição de penalidades para ações não muito desejáveis e de incentivos ao agente em uma ação ótima. Para a expansão do conjunto de episódios ideais pode-se facilitar a atuação do agente, e vice-versa.

Para aprofundar o teste de *ProMon*, a problemática do Mundo de Wumpus também permite a inserção de múltiplos objetivos, que podem ser concorrentes entre si, como por exemplo explorar o máximo possível da caverna labirinto, com o menor número de passos. Essas vertentes mutáveis tenderão a garantir maior diversidade dos testes, mesmo com os conceitos básicos das ideias mantidas para que haja um certo grau de abstração com a aplicação real da ferramenta, o que vai conferir ao teste menor grau de estresse ao projetista e a obtenção de resultados palpáveis.

5.2. Resultados

Como resultados concretos da aplicação de cenários propostos há um retorno satisfatório da atuação de *ProMon* no teste de agentes autônomos.

Isso sugere que a primeira fase de criação de *ProMon*, aqui proposta, é motivadora de uma expansão futura da proposta e da criação de novas ferramentas de testes de agentes racionais.

A geração de episódios ideias, grande desafio desta modelagem, a primeira instância ocorre de forma eficiente, o que é de grande valia pois todo o teste realizado por *ProMon* é embasado na correta identificação destes episódios, por isso a fase de testes do agente *ProMon* é pensada para ser feita sobre o controle do projetista, para que nenhuma peculiaridade deixe de ser explorada. A correta adequação da fase de busca pelos episódios ideias garante a *ProMon* o sucesso como ferramenta de auxílio a testes.

A identificação dos subsistemas falhos deve traz ao projetista uma visão mais focada na correção das falhas que um agente autônomo possa ter e, principalmente, mais eficácia no teste de soluções autônomas por ter maior economia de tempo na busca pelo ponto de origem da falha. As indicações desses pontos de falha são apresentadas de forma didática ao projetista pela saída de testes, e não somente um conjunto de informações, cuja interpretação requeira grande esforço.

Por fim, se torna viável a espera da difusão de *ProMon* como mecanismo de auxílio de teste adotado por um grande número de projetistas de soluções autônomas, uma vez que o uso desta proposta visa, primordialmente, uma maior inserção de soluções baseadas em Inteligência Artificial (IA) no cenário contemporâneo, motivada pela facilidade de implementação e validação dessas soluções e de seu potencial frente aos desafios modernos.

5.3. Conclusão

Vistas as adversidades apresentadas na fase de testes de soluções que empregam o uso de agentes autônomos, esta proposta traz em seu conteúdo, uma alternativa de solução para amenizar os esforços advindos dessa fase de avaliação, tão importante para inserção de uma solução no mercado consumidor.

ProMon como ferramenta de auxílio ao teste de agentes se destaca por sua minúcia frente a origem das possíveis falhas que um agente possa cometer, bem como por sua ampla aplicação, que não reduz o escopo a somente uma tipologia de agente ou de problemática, pelo contrário, proporciona a abertura para a inserção de novos campos ainda inexplorados nesta proposta inicial.

As indicações de falha apresentadas por *ProMon* são propostas de

modo a fornecer o melhor tratamento, feito de forma automática, trazendo ao projetista uma noção de maior utilidade da fase de teste, possibilitando o uso para fins pessoais, de mercado e acadêmico. O que está em sintonia com o objetivo inicial de ampliação dos resultados e minimização dos esforços dos projetistas de soluções inteligentes.

Embora esta proposta possua a primeira vista uma simples definição, o problema para o qual se dispõe trazer uma solução é de grande complexidade, sua sistematização faz com que a proposta seja viável quanto a implementação, afinal não teria acepção alguma em alvitrar algo utópico. Todavia, é necessário sempre estar disposto a testar os bornes de uma solução, e com esta não deve ser diferente, pois as limitações preditas ao decorrer desta proposta podem ser melhor investigadas a medida que novas funções são incorporadas, o que pode por sua vez abre espaço para novos questionamentos sobre a proposta acarretando em novas melhorias.

6. Considerações Finais

Para as considerações finais apresentadas neste capítulo, serão expostas as principais contribuições e limitações desta proposta no que se refere ao teste de agentes racionais, bem como serão exploradas as pretensões de trabalhos futuros. A proposta de *ProMon* será aqui avaliada como passo essencial para sua concepção. Também serão apresentadas as possibilidades vindas por intermédio das limitações iniciais, e a progressão de expansão pensada para a proposta.

Vale lembrar que um agente racional é uma entidade, cujo teste é abstruso, principalmente, por sua característica de autonomia, mas que detém grande potencial para a solução de problemas complexos de forma independente. Mesmo sendo uma tecnologia potencial, ainda há lacunas na formulação de metodologias e técnicas de teste de soluções inteligentes, o que para esta proposta é o grande ponto motivador.

6.1. Contribuições

Por todo esforço empregado para construção e otimização de soluções inteligentes, estas se tornam complexas e por diversas vezes trabalhosas, porém seu potencial como solução para problemas de grande complexidade pode ser aproveitado para proporcionar mais qualidade de vida, para que em um futuro, possivelmente, não tão distante se possa ver a Inteligência Artificial (IA) como grande facilitadora, não apenas no ramo computacional, mas em todas as esferas empregáveis.

Tendo em vista o déficit de soluções que buscam amenizar a problemática enfrentada na fase de testes de sistemas baseados no uso de agentes racionais, esta proposta traz a descrição e modelagem de uma solução potencial, que pode minimizar o esforço do projetista maximizando seus resultados, a fim de trazer soluções, cujo acerto da finalidade de uso seja a maior possível.

Embora esta proposta tenha foco arrestado, a primeira instância, ao teste de agentes das tipologias propostas por Russell e Norvig (2004), seu leque de abertura para maior exploração de novas definições permanece aberto. A expansão se daria de forma facilitada por seu planejamento arquitetural bem definido e pela divisão do trabalho geral do teste em módulos

de atuação.

A contribuição geral desta modelagem já se encontraria na proposta de metodologia de teste, que é utilizada por *ProMon*, pois metodologias e sistematizações são raras de se encontrar para apoio ao teste de agentes racionais e são de grande ajuda ao projetista, principalmente, o que está se iniciando na área. No entanto, a proposta de automação do teste é o real ponto chave deste trabalho, pois traz não somente comodidade para a fase de teste, como possibilita maior produção pela economia de tempo que oferta e pelos aborrecimentos evitados.

A atribuição mestra deste trabalho está na proposta de identificação de episódios ideias para avaliar os episódios de um dado agente de forma a não somente identificar a falha, mas também apontar o subsistema responsável por ela. Tudo isso com possibilidade de aplicação em escopos abertos e fraco aprisionamento a modelos pré-determinados. O que faz classificar os objetivos deste trabalho como alcançados, uma vez que a modelagem de *ProMon* se adequa as lacunas no teste de agentes racionais, porém com objetivos secundários a serem alvejados nas fases que possam vir, posteriormente, a este trabalho, em forma de ampliação.

Portanto esta proposta é enquadrada no campo dos avanços que vem para contribuir e facilitar a vida de projetistas e programadores de agentes racionais, colaborando de forma direta ou indireta para o avanço da tecnologia da informação nas demais áreas do conhecimento, bem como nos demais ambientes de aplicação, gerando benefícios como comodidade e ampliação do poder de trabalho, marcando desta forma mais um salto rumo à era digital.

6.2. Trabalhos Futuros

A continuidade deste trabalho e os aspectos limitantes da proposta são afrontados como uma perspectiva de trabalhos futuros, em que possa ocorrer uma maior abertura de exploração e generalização da solução, apresentando uma codificação da modelagem de *ProMon* para que se possa usufruir dos benefícios ofertados nesta modelagem.

Contudo, se pode alavancar este trabalho com novos horizontes, para que *ProMon* possa atingir maior generalidade de soluções. As restrições de teste para os tipos de agentes aqui adotadas são apenas para delimitar o início desta proposta. Contudo, obtendo bons resultados nesses tipos agentes racionais, os passos que seguem são de ampliação e reorganização da proposta para atender agentes mais complexos.

Por fim, há também o anseio da evolução de *ProMon* de ferramenta

de teste para ferramenta de concepção de agentes, tomando como base sua estrutura fundamental de cópia de um agente testado para a busca de episódios ideais, e a execução de um estudo mais aprofundado dos limites de atuação de *ProMon*.

Referências

BELLIFEMINE, F. L.; CAIRE, G.; GREENWOOD, D. *Developing multi-agent systems with JADE*. [S.l.]: John Wiley & Sons, 2007.

CISCO, C. V. N. The zettabyte era–trends and analysis. *Cisco white paper*, 2016.

COELHO, R. et al. Unit testing in multi-agent systems using mock agents and aspects. In: ACM. *Proceedings of the 2006 international workshop on Software engineering for large-scale multi-agent systems*. [S.l.], 2006. p. 83–90.

DEB, K. et al. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. In: SPRINGER. *International Conference on Parallel Problem Solving From Nature*. [S.l.], 2000. p. 849–858.

DEB, K. et al. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, IEEE, v. 6, n. 2, p. 182–197, 2002.

DURILLO, J. J.; NEBRO, A. J. jmetal: A java framework for multi-objective optimization. *Advances in Engineering Software*, Elsevier, v. 42, n. 10, p. 760–771, 2011.

DURILLO, J. J. et al. jmetal: a java framework for developing multi-objective optimization metaheuristics. *Departamento de Lenguajes y Ciencias de la Computación, University of Málaga, ETSI Informática, Campus de Teatinos, Tech. Rep. ITI-2006-10*, 2006.

HOLLAND, J. H. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. [S.l.]: MIT press, 1992.

MCCORDUCK, P. Machines who think. 2004.

MITCHELL, M. Genetic algorithms: An overview. *Complexity*, Wiley Online Library, v. 1, n. 1, p. 31–39, 1995.

MORENO, M.; PAVÓN, J.; ROSETE, A. Testing in agent oriented methodologies. *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living*, Springer, p. 138–145, 2009.

NGUYEN, C. D. *Testing techniques for software agents*. Tese (Doutorado) — DIT-University, 2008.

NGUYEN, C. D. et al. Evolutionary testing of autonomous software agents. In: INTERNATIONAL FOUNDATION FOR AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS. *Proceedings of The 8th International Conference on Autonomous*

Agents and Multiagent Systems-Volume 1. [S.l.], 2009. p. 521–528.

RUSSELL, S.; NORVIG, P. *Inteligência artificial*. [S.l.]: Elsevier, 2004.

RUSSELL, S.; NORVIG, P. *Inteligência Artificial, 3a Edição*. [S.l.]: Elsevier Brasil, 2014.

SILVEIRA, F. R. d. V. Uma abordagem fundamentada em agentes racionais para o teste de agentes racionais. 2013.

SRINIVAS, N.; DEB, K. Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary computation*, MIT Press, v. 2, n. 3, p. 221–248, 1994.

STOBING, C. How Artificial Intelligence Will Change Our Lives, for Better or Worse. How-To Geek, 2015. disponivel em: https://goo.gl/qu1QTt.

TURING, A. M. Computing machinery and intelligence. *Mind*, JSTOR, v. 59, n. 236, p. 433–460, 1950.

WINIKOFF, M.; CRANEFIELD, S. On the testability of bdi agent systems. *J. Artif. Intell. Res.(JAIR)*, v. 51, p. 71–131, 2014.

WOOLDRIDGE, M. *Intelligent Agents, w: Multiagent Systems-A Modern Approach to Distributed Artificial Intelligence, red. G. Weiss*. [S.l.]: MIT Press, Cambridge, 1999.

YOB, G. Hunt the wumpus. *Creative Computing*, v. 1, n. 5, p. 51–54, 1975.