Uma Abordagem Baseada em Agentes para o Teste Caixa Preta de Agentes Racionais: Um Estudo de Caso com Agentes Reativos

Fca. Raquel de V. Silveira, Gustavo Augusto L. de Campos, Mariela I. Cortés

Universidade Estadual do Ceará (UECE)
Caixa Postal 60.740-903 – Fortaleza – CE – Brasil

raquel_silveira@ifce.edu.br, {gustavo, mariela}@larces.uece.br

Abstract. *In the theoretical references available to guide the design of rational agents, there are few test techniques to validate them. It is known that this validation depends on the selected test cases, which should arrange for information concerning the components of the agent tested that are underperforming. This paper proposes an approach that aims to contribute to the process of testing these programs through of the Thestes agent which solves the problem of selecting test case and the ProMon agent which does the diagnostic of the issues on the tested agent. The first experiments aimed to evaluate the approach by selecting test cases for simple and with internal state reactive agent in partially observable environment.*

Resumo. *Nos referenciais disponíveis para orientar o projeto de agentes existem poucas técnicas de testes para validá-los. Sabe-se que essa validação depende dos casos de teste selecionados, os quais devem gerar informações que identifiquem os componentes do agente testado que estão causando o desempenho insatisfatório. Este trabalho propõe uma abordagem que visa contribuir com o teste destes programas através do agente Thestes que resolve o problema de seleção de casos de teste e do agente ProMon que realiza o diagnóstico das falhas do agente testado. Os primeiros experimentos visaram avaliar a abordagem selecionando casos de teste para programas de agentes reativos simples e com estado interno em ambiente parcialmente observável.*

1. Introdução

Agente é uma entidade capaz de perceber seu ambiente por meio de sensores e agir nesse ambiente por intermédio de atuadores. O comportamento de um agente pode ser descrito por uma função do agente capaz de mapear qualquer sequência de percepções específica para uma ação. Esta função do agente é implementada concretamente pelo programa do agente, que é executado em uma arquitetura adequada. Idealmente, os agentes racionais devem agir visando alcançar o melhor resultado esperado, avaliado de acordo com uma medida de desempenho [Russell e Norvig 2013].

Uma vez que sistemas baseados em agentes estão cada vez mais assumindo diversas áreas como, a assistência ao paciente, simulação de campo de batalha, detecção de intrusão, jogos, etc., garantias do correto funcionamento desses sistemas precisam ser dadas aos usuários. Isto exige uma investigação de estruturas de engenharia de software, incluindo requisitos de engenharia, arquitetura e técnicas de teste, para proporcionar processos adequados de desenvolvimento e verificação de software [Nguyen et al. 2012].

Apesar de existirem alguns esforços no sentido de facilitar o desenvolvimento de sistemas baseados em agentes, pouco tem sido feito na direção de propor métodos e técnicas para testar a corretude destes sistemas [Nguyen et al. 2012]. Para a realização de testes em agentes racionais, é necessário que as técnicas existentes para testes de software tradicionais sejam adaptadas e combinadas visando a detecção de diferentes falhas, tornando os agentes de software mais confiáveis [Houhamdi 2011].

Um dos motivos possíveis para ausência de técnicas de testes de agentes é a difícil aplicação de técnicas que sejam capazes de garantir a confiabilidade destes sistemas, devido as propriedades peculiares e a natureza específica dos agentes de software, que são projetados para serem distribuídos, autônomos e deliberativos, o que pode criar o efeito da irreprodutividade, que significa não ser possível garantir que duas execuções do sistema levem ao mesmo estado, mesmo se as mesmas entradas forem utilizadas. Como consequência, procurar um erro específico pode ser difícil, já que não é possível reproduzi-lo a cada nova execução [Nguyen et al. 2012].

A realização de testes de software tradicionais, que possuem entradas e saídas previsíveis, não é uma atividade trivial. Testar agentes autônomos é um desafio, uma vez que a execução de ações é baseada nas decisões do agente, que podem ser diferentes da perspectiva do usuário, embora sendo apropriada [Nguyen et al. 2012; Silveira et al. 2013].

Este trabalho apresenta uma abordagem que visa contribuir com o processo de testes caixa preta de agentes racionais. Os pressupostos são que qualquer tipo de teste depende dos casos de teste selecionados, os quais devem gerar informações que permitam identificar os componentes na estrutura do programa do agente artificial testado estão causando desempenho insatisfatório. Considerando que nem sempre os melhores casos estão disponíveis a priori e, dependendo do ambiente de tarefa do agente, pode existir uma grande quantidade de casos a serem observados, a seleção de um conjunto de casos de teste é um problema de busca em um espaço de estados composto por uma grande família de conjuntos de casos possíveis.

Assim, a abordagem proposta consiste em uma formulação matemática para o problema de seleção de casos de teste de agentes; em um agente orientado por utilidade que emprega aspectos presentes em metaheurísticas baseadas em população para encontrar conjuntos de casos de teste satisfatórios, isto é, descrições de ambientes específicos em que as histórias associadas do programa de agente testado em seu ambiente obtiveram baixo desempenho; e um agente de monitoramento que gere para o projetista informações relevantes sobre o desempenho e as falhas do agente durante os testes, permitindo realizar melhorias nos programas agentes.

O presente trabalho está organizado da seguinte forma: a Seção 2 discorre sobre os principais conceitos relacionados aos agentes racionais e aos testes de agentes. Os trabalhos relacionados que consideram os conceitos de testes de agentes autônomos são apresentados na Seção 3. As seções 4 e 5 descrevem, respectivamente, a abordagem proposta e um estudo de caso que ilustra o funcionamento e uma avaliação da abordagem. Ao final, conclusões e trabalhos futuros são conduzidos na Seção 6.

2. Referencial Teórico

Esta seção fornece um referencial teórico sobre as áreas relacionadas ao escopo deste trabalho. Inicialmente, são contextualizados os agentes racionais. Em seguida, são

contextualizados os testes de agentes, realizando uma analogia com os testes de software tradicionais.

2.1. Agentes Racionais

O agente racional seleciona suas ações objetivando o melhor resultado possível, ou na presença de incertezas, o melhor resultado esperado conforme a medida de desempenho estabelecida para avaliar seu comportamento. Conceber agentes racionais para atuar em ambientes de tarefa complexos é uma tarefa não trivial [Russell e Norvig 2013].

O trabalho da Inteligência Artificial consiste em projetar o programa do agente, que implementa a função do agente e será executado em alguma arquitetura, ou seja, um dispositivo de computação com atuadores e sensores. Dependendo do ambiente, o projeto do agente pode ser idealizado considerando quatro tipos básicos de programas de agentes: (1) agentes reativos simples (selecionam ações com base na percepção atual, ignorando o histórico de percepções), (2) agentes reativos baseados em modelos (o agente mantém um estado interno que depende do histórico das percepções), (3) agentes baseados em objetivos (além do estado atual, mantém informação sobre os objetivos que descrevem situações desejáveis); (4) agentes baseados em utilidade (possuem uma função utilidade que mapeia um estado em um grau de felicidade associado). Em ambientes onde o agente não conhece os estados possíveis, nem os efeitos das suas ações, a concepção de um agente racional pode requisitar de um programa de agente com capacidade de aprendizagem [Russell e Norvig 2013].

Os quatro tipos de programas agentes podem ser subdivididos em três subsistemas de processamento de informação principais. O primeiro, o subsistema de percepção, mapeia uma informação sobre percepção (P) em uma representação abstrata (Estado) útil para o agente, **ver**: P → Estado. O segundo, o subsistema de atualização de estado interno, mapeia a representação da percepção atual e a informação sobre o estado interno (EI) mantido pelo agente em um novo estado interno, **próximo**: Estado x EI → EI. Por último, o subsistema de tomada de decisão, mapeia uma informação sobre estado interno em uma ação possível (A), **ação**: EI → A. Por exemplo, para o caso do programa agente reativo simples, a função ação seleciona ações baseando-se na percepção atual, mapeada pela função **ver**, e em um conjunto de regras no formato condição-ação [Wooldridge 2002].

2.2. Testes de Agentes

Teste de software é uma atividade que tem como objetivo avaliar a qualidade do produto e melhorá-la através da identificação de defeitos e problemas. Um teste bem-sucedido para detecção de defeitos é aquele que faz com que o sistema opere incorretamente e, como consequência, expõe o defeito existente [Sommerville 2011].

Qualquer produto que passe por engenharia pode ser testado por uma das duas maneiras: fazendo-se um exame interno do detalhe procedimental (caixa branca) e examinando os aspectos funcionais do sistema através da interface, sem se preocupar com a estrutura interna (caixa preta). Para os testes caixa branca e preta, diferentes níveis de testes podem ser usados. Os principais níveis de teste aplicáveis aos produtos de engenharia são: unitário para explorar a menor unidade do projeto; integração para testar a integração dos módulos; sistema para avaliar o software em busca de falhas; e

aceitação para simular as operações de rotina do sistema com os usuários finais [Pressman e Maxim 2014; Rocha et al. 2001].

Devido as propriedades peculiares dos agentes racionais (propriedades reativas, de memória, orientação por metas e por utilidade, e de aprendizagem) e de seus ambientes de tarefa, existe uma procura por novas técnicas de testes relacionadas anatureza particular dos agentes. Para a realização de testes em agentes inteligentes, é necessário que as técnicas existentes para testes de software sejam adaptadas e combinadas visando a detecção de diferentes falhas, tornando os agentes de software mais confiáveis. A maioria dos trabalhos da literatura consiste em adaptações das técnicas de testes de software convencional. No caso dos agentes racionais, sabe-se que estas adaptações devem buscar avaliar a racionalidade das ações e dos planos executados pelo agente em seu ambiente de tarefa [Houhamdi 2011].

Algumas abordagens focam a produção de artefatos de teste para dar suporte às metodologias de desenvolvimento de sistemas de agentes [Nguyen 2008]. O pressuposto na maioria dos trabalhos é que uma boa avaliação do agente depende dos casos de testes selecionados. Bons casos de teste devem providenciar a geração de informações sobre o desempenho insatisfatório dos componentes na estrutura do agente e do funcionamento destes componentes de maneira integrada [Zina 2011].

Na adaptação dos testes tradicionais para o teste de agentes, a literatura classifica os testes de agentes nos níveis: unitário (testa as unidades que compõem o agente), agente (testa a integração de diferentes módulos dentro do agente), integração (testa a integração do agente com os recursos compartilhados), sistema (testa as propriedades que o sistema deverá atingir) e aceitação (testa o agente no ambiente quanto ao atendimento das especificações do cliente) [Nguyen 2008].

3. Trabalhos Relacionados

Na literatura de engenharia de software orientada a agentes, pesquisas têm sido realizadas sobre os diferentes aspectos relacionados ao teste de agentes. Cada uma das quais com diferentes abordagens e perspectivas. Entretanto, o teste de agente é uma atividade desafiadora e um processo de teste estruturado para agentes ainda é requerido [Houhamdi 2011].

Na abordagem de teste para agentes apresentada em Houhamdi (2011) é especificado um processo estruturado para a geração de testes que complementa a metodologia Tropos [Mylopoulos e Castro 2000] e reforça a relação mútua entre a análise de objetivos e testes. O processo fornece uma forma de derivar casos de teste a partir dos objetivos dos agentes. Essa abordagem não contempla: a noção de agentes racionais, uma medida de avaliação de desempenho e uma simulação que possa monitorar o comportamento do agente ao executar as ações para realizar seus objetivos.

Nguyen (2008) propõe um método de análise orientada a objetivos, visando um processo de testes sistemático e abrangente para agentes que engloba o processo de desenvolvimento do agente de acordo com a metodologia Tropos [Mylopoulos e Castro 2000] a partir dos requisitos iniciais até a implantação. Nguyen (2008) propõe uma metodologia de produção de artefatos de testes a partir das especificações dos agentes e do *design*, e usa estes artefatos para detectar problemas. Os casos de teste são gerados automaticamente e evoluem guiados por mutação e função de qualidade.

Uma abordagem evolucionária para a realização dos testes de agentes autônomos é proposta em Nguyen et al. (2012) e aplica o recrutamento dos melhores casos de teste para evoluir os agentes. Para cada agente é dado um período experimental em que o número de testes com diferentes níveis de dificuldade são executados. Os agentes são recrutados apenas quando passam pelo critério de qualidade. Tanto em Nguyen (2008) quanto em Nguyen et al. (2012) o agente a ser testado é implementado de acordo com o modelo BDI (*Belief*, *Desire* e *Intention*). Apesar de considerar a noção de agentes racionais, a abordagem não emprega esquemas de simulação para monitorar o comportamento do agente ao executar as ações para realizar seus objetivos.

4. Abordagem Proposta

A abordagem proposta para o teste de agentes racionais está fundamentada, principalmente, na noção de agentes racionais e na utilização de casos de teste gerados de acordo com os objetivos na medida de avaliação de desempenho do agente testado. Além disso, é considerada a simulação da avaliação de desempenho das interações do agente testado com seu ambiente e estratégias de busca local multiobjetivo para encontrar casos de teste e histórias correspondentes em que o agente não foi bem avaliado.

4.1. Visão Geral da Abordagem

A abordagem (Figura 1) considera que, além do projetista do agente testado, existem quatro programas de agentes envolvidos: o programa do agente a ser testado, Agent, concebido pelo projetista; um programa de agente ambiente de tarefa, Amb; um programa de agente para a seleção de casos de testes, Thestes; e um programa agente monitorador, ProMon, que recebe dados de Thestes e disponibiliza para o projetista informações sobre o desempenho e sobre as falhas de Agent.

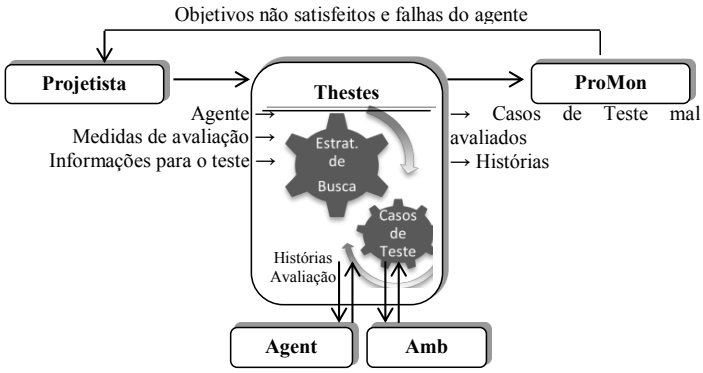


Figura 1. Visão geral da abordagem

O projetista é a pessoa responsável por conceber o programa do agente artificial Agent, a medida de avaliação de desempenho e alimentar outras informações necessárias para o agente Thestes iniciar o processo de teste de Agent em Amb. O agente Thestes consiste em um agente de resolução de problemas de seleção de casos de teste que realiza busca local no espaço de estados de casos de teste orientado por utilidade. Este agente envia para o agente ProMon um conjunto de soluções eficientes determinada pela estratégia de busca multiobjetivo, ou seja, descrevendo casos de teste em que Agent possui o comportamento mais inadequado, um conjunto de histórias correspondentes e seus valores de utilidade. ProMon conhece o tipo de programa de

agente sobre o qual Agent foi concebido, ao receber as informações enviadas por Thestes, identifica para o projetista: os objetivos na medida de avaliação que não estão sendo satisfeitos adequadamente por Agent, os episódios nas histórias de Agent em Amb que são falhas, os episódios ideais correspondentes, e os módulos de processamento da informação de Agent (**ver**, **próximo**, **ação**) que estão gerando as falhas. As informações sobre desempenho permitem que o projetista avalie a racionalidade das ações e planos (soluções) que o agente selecionou ao longo das interações que manteve com o ambiente de tarefa.

4.2. Problema de Seleção de Casos de Teste de Agentes

Um agente é racional se for capaz de tomar decisões corretas, ou seja, que realizem seus objetivos em um ambiente de tarefa, ou, quando não for possível realizar todos os objetivos, tomar as decisões de maior sucesso, definido de acordo com alguma medida de avaliação de desempenho. Esta medida considera um ou mais aspectos do ambiente.

A Tabela 1 especifica uma medida de avaliação de desempenho para o caso de um programa aspirador de pó (uma versão adaptada de Russell e Norvig (2013)) que deve limpar um ambiente e maximizar os níveis de limpeza do ambiente e de energia em sua bateria ao final da tarefa. A primeira coluna descreve parte da percepção do agente (sala limpa = L ou sala suja = S) em cada episódio possível. A segunda coluna descreve as ações possíveis nesses episódios (aspirar = Asp, direita = Dir, esquerda = Esq, acima = Ac, abaixo = Ab e não operar = N-op). A terceira e quarta colunas são a duas funções escalares (av_E e av_L) para medir o desempenho do agente em cada episódio nos objetivos energia e limpeza, respectivamente. A última coluna destaca os episódios com comportamento inadequado, provavelmente por uma falha na função **ver** e/ou no conjunto de regras condição-ação nas funções **ação** do programa agente aspirador de pó. O projetista do agente deve considerar este tipo de medida e conceber o programa visando maximizar seu desempenho nas histórias do agente no ambiente.

Tabela 1. Medida de avaliação desempenho

$\mathbf{P^K}$	$\mathbf{A^K}$	$\mathbf{av_E(P^K, A^K)}$	$\mathbf{av_L(P^K, A^K)}$	**Falha**
..., L, ...	Asp	-1.0	0.0	X
..., L, ...	Dir, Esq, Ac, Ab	-2.0	1.0	
..., L, ...	N-op	0.0	0.0	
..., S, ...	Asp	-1.0	2.0	
..., S, ...	Dir, Esq, Ac, Ab	-2.0	-1.0	X
..., S, ...	N-op	0.0	-1.0	X

A abordagem proposta aplica a modelagem do problema de seleção de casos de teste realizada por Silveira et al. (2014). Os autores consideram que, se o programa agente for inadequado ao seu ambiente, as funções objetivo de inadequação, ou seja, as funções escalares na medida de avaliação modificadas pelo sinal de menos (-), serão maximizadas. Entretanto, pode não existir um conjunto que seja ótimo em todas as funções e, neste caso, a tarefa consiste em encontrar um conjunto de casos de teste (CasosTEST) satisfatório, ou seja, o desempenho do programa de agente no ambiente é insatisfatório e que permita ao projetista perceber as propriedades limitativas do agente.

Supondo-se que a função Utilidade é preferencialmente independente, ou seja, o grau de utilidade de um objetivo independe dos valores assumidos pelos demais, esta primeira concretização incorporou em Thestes uma função Utilidade no formato linear:

$$\text{Utilidade}(f_{inad}(H(\text{CasosTEST}))) = -\sum_{m=1}^{M} w_m * f_m(H(\text{CasosTEST}))$$

tal que $w_m \geq 0$ e representa o peso atribuído a cada objetivo, m = 1, ..., M. H(CasosTEST) representa o conjunto de histórias de comprimento NInt de Agent em Amb. Assim, o problema de seleção de casos de teste é definido como:

'maximizar' Utilidade(f_{inad}(H(CasosTEST)))

sujeito a: CasosTEST ∈ P(Ω) e

H(CasosTEST) ∈ P((PxA)NInt)

4.3 O Agente Thestes

A noção de agentes é uma ferramenta disponível bastante útil para analisar e conceber sistemas e foi empregada na concepção do sistema de resolução do problema de seleção de casos testes. Do ponto de vista de projetos de agentes únicos, a literatura sobre o assunto disponibiliza diversas arquiteturas abstratas de agentes e diversos tipos de programas de agentes que podem ser concretizados para resolver problemas em ambientes de tarefas que requisitam uma tomada de decisão não trivial. O agente Thestes foi concebido como um agente de resolução de problemas de seleção de caso de testes. Seu esqueleto fundamenta-se na estrutura do programa de agentes orientado por utilidade especificada por Russell e Norvig (2013) e na arquitetura abstrata do agente com estado interno Wooldridge (2002).

A Figura 2 ilustra a estrutura do agente de resolução de problemas Thestes. Seu subsistema de percepção, **ver**, mapeia as informações necessárias ao teste de Agent em uma representação computacional, Estado^K, adequada ao processamento dos outros subsistemas: (Agent, Amb, ParâmetrosBusca, ParâmetrosSimulação). O subsistema de atualização de estado interno, **próximo**, armazena as informações em Estado^K, considera os parâmetros em ParâmetrosSimulação e gera um conjunto inicial CasosTEST de maneira aleatória. Considerando o estado interno atualizado, a função **ação** de Thestes inicia um processo de busca local baseada em populações e orientada por uma função utilidade visando encontrar uma ação satisfatória Ação^K para ser enviada ao projetista ou seja, descrições de ambientes específicos em que as histórias associadas de Agent em Amb possuem baixo desempenho.

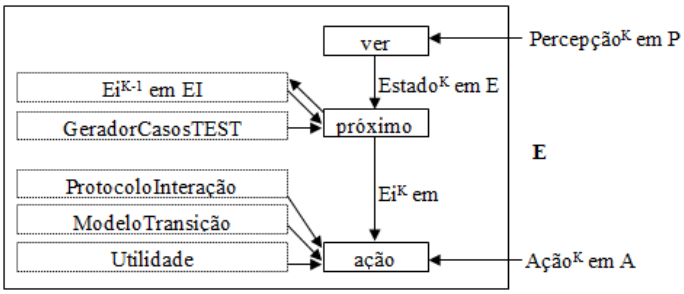


Figura 2. Estrutura do programa agente Thestes

A função **ação** utiliza informações de um modelo de transição de estados para gerar novos casos de teste a partir de CasosTEST, e o protocolo de interação e a função utilidade para, respectivamente, obter as histórias correspondentes aos casos de teste e

avaliar o desempenho de Agent nestas histórias. O modelo de transição modifica os conjuntos de casos de teste, considerando os casos de teste em uma solução corrente, Pop^t = CasosTEST, os valores de desempenho correspondentes, medidos por uma função Utilidade para gerar novos casos de teste, Pop^{t+1}. O modelo de transição genérico considera os NCasos em um conjunto Pop^t corrente e escolhe: (a) os casos de teste que serão modificados, e (b) as mudanças que serão realizadas nestes casos.

Thestes conhece Agent e Amb, bem como o protocolo ProtocoloInteração. A função **ação** considera estas informações e ParâmetrosSimulação no processo de tomada de decisão. Assim, foi concebido um mecanismo de interação que simula as interações entre Agent e Amb, quando Amb é inicializado com informações de um caso de teste ($Caso_i \in$ CasosTEST) e anota os episódios ($Ep^p(h(Caso_i)) \in$ PxA) da história ($h(Caso_i) \in$ (PxA)NInt). Thestes intercepta as mensagens de Agent e Amb, a fim de obter as histórias dos casos de teste. A Figura 3 ilustra o funcionamento deste mecanismo.

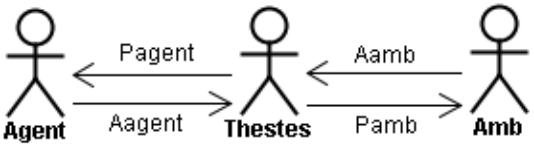


Figura 3. Simulação das interações Agent-Amb

O mecanismo alimenta Amb com informações em P_{Amb} a respeito de um caso de teste específico pertencente ao conjunto solução corrente CasosTEST. O programa Amb armazena essas informações e envia em A_{Amb} as informações de estado corrente para Agent. Antes de repassar estas informações para Agent, o mecanismo anota estas informações. Ao perceber estas informações em P_{Agent}, Agent processa-as e seleciona uma ação que é enviada em A_{Agent} para o mecanismo. Ao receber estas informações sobre ação, o mecanismo encerra a anotação de um episódio da história, envia em P_{Amb} as informações sobre ação para Amb e inicia outro ciclo de interação, que deve ser repetido até a anotação de uma história completa.

4.4 O Agente ProMon

O agente ProMon recebe de Thestes três informações importantes em $Ação^K$: (1) um conjunto CasosTEST contendo os casos de teste onde Agent teve um comportamento inadequado, (2) as histórias de Agent em Amb e (3) 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 Agent falhou e os episódios ideais correspondentes, e (2) uma identificação do tipo de falha, indicando o subsistema de processamento de informação de Agent que ele presume esteja causando a falha, ou seja: (a) subsistema de percepção, **ver**, (b) subsistema de atualização de estado interno, **próximo**, e (c) subsistema de tomada de decisão, **ação**.

O agente ProMon foi concebido como um agente reativo baseado em modelos. Mais especificamente o processo de monitoramento e diagnóstico de falha realizado pelo agente ProMon foi 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 em CasosTEST H(CasosTEST) e identifica todos os episódios contendo falhas nessas

histórias. Este trabalho coloca em destaque apenas o subsistema **próximo** de ProMon para identificação dos episódios que são falhas.

Assim, como no caso do agente Thestes, esta função considera o protocolo ProtocoloInteração, o programa ambiente Amb e uma versão totalmente observável do agente testado, denotada por Agent* para identificar se um episódio em uma interação *t*, em uma história associada a um $Caso_i$ em CasosTEST, e gerar dois conjuntos de episódios: o conjunto de episódios ideais na interação, Ep_{ideais}^{t}, e o conjunto de episódios com falhas nas histórias associadas aos casos em CasosTEST, Ep_{falhas}.

Assim, de acordo com a notação empregada na formulação do problema de seleção de casos de teste do agente Thestes, seja:

- **Agent*:** versão totalmente observável do programa Agent a ser testado.
- **$h^*(Caso_i) \in (PxA)^{NInt}$:** história de comprimento NInt de Agent* em Amb correspondente ao $Caso_i \in$ CasosTEST.
- **$Ep^t(h^*(Caso_i)) \in PxA$:** ou, mais especificamente, $Ep((P^t, A^t*))) \in PxA$ um episódio na interação *t* da história de Agent* em Amb correspondente ao caso $Caso_i \in$ CasosTEST.

O conjunto de episódios ideais em uma interação *t*, Ep_{ideais}^{t}, é formado por todos os episódios possíveis de serem produzidos por Agent* na interação, $Ep((P^t, A^t*))$, que satisfazem pelo menos uma das duas condições abaixo:

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

$av_m(Ep((P^t, A^t*))) \geq av_m(Ep((P^t, A^t)))$;

(2) A^t* é melhor que ou é equivalente a A^t considerando o ponto de vista do projetista.

E, consequentemente, episódios $Ep((P^t, A^t))$, produzidos pelo agente testado *Agent* que não pertencem ao conjunto Ep_{ideais}^{t}, devem compor o conjunto de episódios com falha Ep_{falhas}.

A condição (2) depende do ponto de vista do projetista. Ela foi introduzida visando identificar falhas que não são percebidas diretamente na especificação da medida de avaliação de desempenho. Por exemplo, considerando a medida de avaliação de desempenho para um programa aspirador de pó descrita na Tabela 1, é possível perceber que as linhas 1, 5 e 6 identificam episódios que são falhas. Entretanto, a Tabela não identifica episódios que são falhas em função do agente movimentar-se para salas vizinhas que não estão sujas ou retornar desnecessariamente às salas que já foram visitadas. Assim, diferentemente da condição (1), que considera a medida de avaliação de desempenho, a condição (2) deverá ser especificada de acordo com o domínio de aplicação do agente testado Agent.

Estas são as condições no antecedente das regras condição-ação do agente ProMon, que, quando satisfeitas, sugerem para o projetista o tipo de análise que ele deve realizar de maneira a identificar melhor o módulo de processamento de informação de Agent que está causando a falha.

5. Avaliando a Abordagem

Nesta seção é ilustrado o funcionamento de Thestes e de ProMon, resolvendo um problema de seleção para testar um agente aspirador de pó em dois programas agentes: reativo simples e reativo com estado interno, ambos com percepção local e avaliados em um ambiente com várias salas considerando os atributos energia e limpeza (Tabela 1).

5.1. O Ambiente de Tarefa e o Agente em Teste

A tarefa de Thestes consiste em selecionar um conjunto de ambientes que sejam satisfatórios para testar um programa agente aspirador de pó com regras condição-ação, enquanto ProMon realiza a identificação dos episódios que são falhas. Um ambiente difere de outro quanto à localização e à quantidade de salas sujas. Todo ambiente é parcialmente observável, ou seja, pressupõe-se que o aspirador percebe o ambiente, mas sua função ver consegue mapear apenas o estado da sala em que o agente está.

O programa agente aspirador de pó reativo simples (Asp_RS_Parcial) foca na seleção das ações com base na percepção atual, ignorando o histórico de percepções, em um ambiente parcialmente observável, ou seja, a função ver permite perceber apenas o estado, sujo ou limpo, da sala em que o agente está. Assim, este primeiro programa apresenta de uma maneira simples mas útil, uma primeira ilustração e avaliação da abordagem. A Figura 4 apresenta as regras condição-ação desse agente.

se estado da sala é S **então faça** Asp
se estado da sala é L **então faça** movimento aleatório (Ac, Esq, Dir, Ab)

Figura 4. Regras condição-ação do agente aspirador reativo simples

O segundo programa agente testado foi concebido de acordo com a estrutura do agente reativo com estado interno (Asp_REi_Parcial) considerando que o ambiente é parcialmente observável. Este agente possui um estado interno que armazena o histórico das percepções obtidas e uma ação é selecionada levando em considerando esta informação. A Figura 5 apresenta as regras condição-ação desse agente.

se estado da sala é S **então faça** a ação Asp
se estado da sala é L e NaoVisitou(norte) **então faça** a ação Ac
se estado da sala é L e NaoVisitou(sul) **então faça** a ação Ab
se estado da sala é L e NaoVisitou(leste) **então faça** a ação Dir
se estado da sala é L e NaoVisitou(oeste) **então faça** a ação Esq
se estado da sala é L e visitou todas **então faça** uma ação aleatória

Figura 5. Regras condição-ação do agente aspirador com estado interno

5.2. Concretização de Thestes para o Problema

O esqueleto de Thestes pressupõe a existência de um conjunto CasosTEST corrente contendo NCasos de teste, que é uma solução inicial gerada pela função **próximo**. A função **ação** do agente considera quatro componentes principais: (1) um ModeloTransição de estados que opera sobre CasosTEST para gerar novos conjuntos, (2) uma função utilidade para avaliar os conjuntos gerados, (3) uma estratégia para selecionar entre os conjuntos avaliados um novo conjunto corrente mais útil, e (4) um teste para o novo conjunto CasosTEST. Para os componentes (1), (3) e (4) o estudo empregou noções de SBSE (*Search-Based Software Engineering*), mais especificamente, nos experimentos foram utilizados os algoritmos genéticos (GA)

[Holland 1975]. Para o componente (2), empregou uma função Utilidade no formato linear, conforme descrito na Seção 4.2.

No contexto do GA, CasosTEST, contendo as descrições de ambientes de tarefa compostos de *n x n* salas, foi representado por uma população, onde cada indivíduo codifica um ambiente e cada gene codifica o estado da sala, em termos de sujeira. O ModeloTransição baseado em GA considera CasosTEST como uma população que está apta a passar pelas etapas de seleção de pares (empregado o método da roleta), cruzamento e mutação, que provocam a evolução e que permitem à função ação realizar simulações, avaliar a utilidade dos indivíduos e compor uma nova população melhor. Visando prevenir a perda do melhor caso de teste encontrado empregou-se elitismo nas modificações do conjunto CasosTEST. O melhor caso de teste é aquele onde o agente apresenta o pior desempenho na avaliação.

5.3. Experimentos

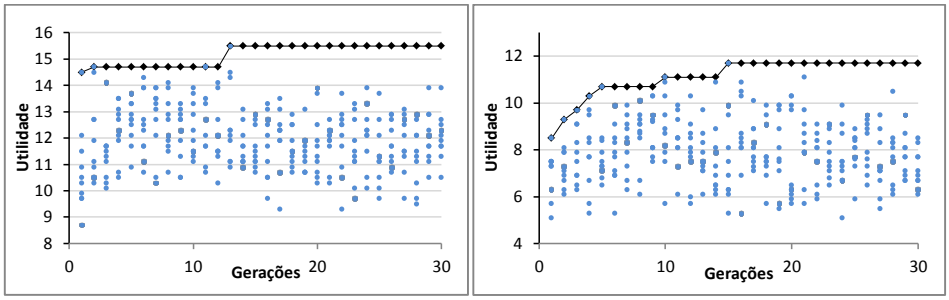
Esta seção apresenta os experimentos realizados para o teste do agente aspirador de pó. ParâmetrosBusca descrevem o operador e a probabilidade de cruzamento (O_{Cros} e P_{Cros}) entre os casos de teste, o operador e a probabilidade de mutação (O_{Mut} e P_{Mut}), o número máximo de execuções ($K_{máx}$), e o valor de utilidade máxima que pode ser alcançada por uma história ($U_{máx}$). As informações sobre $K_{máx}$ e $U_{máx}$ definem a condição de parada no mecanismo de teste. As informações em ParâmetrosSimulação descrevem a quantidade (NCasos) e o tamanho (n^2) de ambientes em CasosTEST, o número máximo de interações entre Agent e Amb em qualquer simulação (N_{int}), ou seja, o número máximo de episódios em cada história, e o número de simulações realizadas em um mesmo ambiente (Ns). A Tabela 2 apresenta estas informações.

Tabela 2. Informações em ParâmetrosBusca e ParâmetrosSimulação

ParâmetrosBusca						**ParâmetrosSimulação**			
O_{Cros}	P_{Cros}	O_{Mut}	T_{Mut}	$K_{máx}$	$U_{máx}$	NCasos	n^2	N_{int}	Ns
SinglePointCrossover	0,9	*Uniform*	0,6	30	100000	10	25	25	5

O número máximo de interações (N_{int}) de Agent com Amb, a serem simuladas em cada um dos 10 casos (NCasos) na população, é 25. O valor de utilidade máxima ($U_{máx}$) é alto, e a condição de parada no mecanismo de teste da estratégia de busca é definida considerando 30 ciclos de execuções da função ação ($K_{máx}$). Para cada caso de teste na população foram realizadas cinco simulações (Ns). A Figura 6 apresenta a função Utilidade no formato linear com pesos iguais para limpeza e energia, respectivamente, na medida de avaliação, ou seja, $w_L = w_E = 0,5$ para o agente Asp_RS_Parcial em (a) e Asp_REi_Parcial em (b). Os pontos não lineares representam os casos de teste em CasosTEST em cada geração. Os pontos lineares identificam o melhor caso de teste na geração.

No caso do agente reativo simples com observabilidade parcial (Figura 6 (a)), o melhor caso de teste foi obtido na décima quarta geração, com valor de utilidade igual a 15,5. Este caso serviu como referencial para os casos em CasosTEST nas gerações posteriores com valores de utilidade menores que o melhor. No caso do agente reativo com estado interno (Figura 6 (b)), o melhor caso de teste foi obtido na décima quinta geração, com valor de utilidade igual a 11,7.

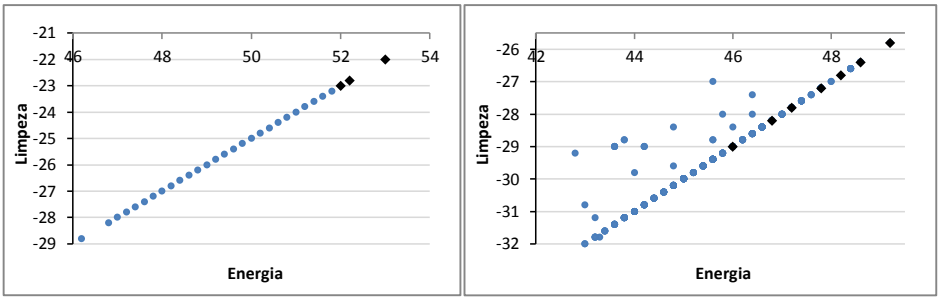


(a) Asp_RS_Parcial (b) Asp_REi_Parcial

Figura 6. Valores de utilidade de 10 casos CasosTEST em 30 gerações

Considerando os valores de utilidade dos melhores casos de teste, Thestes detectou que o agente mais inadequado para o ambiente parcialmente observável é o reativo simples, quando comparado com o agente reativo com estado interno. Neste caso, a anotação das salas que foram visitadas pelo subsistema de atualização de estado interno (**próximo**) do agente baseado em modelos possibilitou a concepção de um subsistema de tomada de decisão (ação) mais refinado para o agente reativo com estado interno. O conjunto de regras componentes deste subsistema evitou o retorno desnecessário às salas que já visitadas em interações anteriores com o ambiente, o que não foi possível para o agente reativo simples.

A Figura 7 apresenta os valores das funções de inadequação de limpeza e energia associados aos 10 casos em CasosTEST nas 30 gerações. Comparando os resultados entre os agentes, considerando o caso de teste em CasosTEST com maior valor de utilidade conforme Figura 6, é possível observar que para ambos os atributos, energia e limpeza, apresentados na Figura 7, o agente reativo simples com observabilidade parcial possui um comportamento mais inadequado, com $-f_E = 53,0$ e $-f_L = -22,0$, enquanto o agente reativo com estado interno possui comportamento menos inadequado, com $-f_E = 49,2$ e $-f_L = -25,8$. Isto pode ser justificado devido às propriedades internas de cada um dos agentes.



(a) Asp_RS_Parcial (b) Asp_REi_Parcial

Figura 7. Valores de Inadequação de Limpeza e Energia dos 300 casos

A princípio, como os dois objetivos na medida de avaliação de desempenho tem o mesmo valor de importância ($w_L = w_E = 0,5$) na função utilidade, a abordagem privilegia os casos em que Agent tem um comportamento mais inadequado em termos do consumo de energia que em termos de limpeza, quase que maximizando este nível de inadequação conforme indicado em todas as gerações. Isto se justifica, pois a medida de avaliação de desempenho pontua negativamente em termos de energia todos os

episódios possíveis para Agent, devido ao agente sempre gastar energia ao executar suas ações, e positivamente em termos de limpeza os episódios. A Tabela 3 ilustra cinco episódios de uma simulação da interação de Agent em Amb, no ambiente que obteve melhor valor médio de utilidade para o agente reativo simples.

Tabela 3. História parcial de Agent em Amb

K	P^K	A^K	$-av_E(P^K, A^K)$	$-av_L(P^K, A^K)$
1	..., L, ...	Ab	2.0	-1.0
2	..., L, ...	Dir	2.0	-1.0
3	..., L, ...	Ab	2.0	-1.0
4	..., S, ...	Asp	1.0	-2.0
5	..., L, ...	Esq	2.0	-1.0

O ambiente de tarefa selecionado foi constituído de 25 salas com a seguinte configuração: [[L, L, L, L, S], [L, L, S, S, L], [L, S, S, S, S], [L, L, L, S, S], [L, L, S, S, S]]. O valor de utilidade foi U = 15,5 e os valores de inadequação: $-f_E = 49{,}0$ e $-f_L = -26{,}0$. Os demais episódios da história seguem o mesmo padrão. Conforme esperado, o agente aspirador é mais adequado ao ambiente considerando o critério de limpeza que o critério de energia. Uma análise breve das regras condição-ação do aspirador de pó confirmará esta proposição. A história obtida por Asp_REi_Parcial segue o mesmo padrão. A Tabela 4 ilustra o diagnóstico de ProMon a partir dos episódios demonstrados na Tabela 3 para o Asp_RS_Parcial. A tabela apresenta em $Ep_{ideais}{}^K$ os episódios ideais gerados por Agent* e os episódios que são falha identificados por ProMon, em Ep_{falha}.

Tabela 4. História parcial de Asp_RS_Parcial em Amb

K	Ep^K				$Ep_{ideais}{}^K$	Ep_{falha}
	P^K	A^K	$-av_E(P^K, A^K)$	$-av_L(P^K, A^K)$		
1	..., L, ...	Ab	2.0	-1.0	(...,L,...; Ab), (...,L,...; Dir)	
2	..., L, ...	Dir	2.0	-1.0	(...,L,...; Ab), (...,L,...; Dir)	
3	..., L, ...	Ab	2.0	-1.0	(...,L,...; Ab), (...,L,...; Dir)	
4	..., S, ...	Asp	1.0	-2.0	(...,S,...; Asp)	
5	..., L, ...	Esq	2.0	-1.0	(...,L,...; Dir)	X
6	..., L, ...	Ab	2.0	-1.0	(...,L,...; Dir)	X
7	..., L,...	Dir	2.0	-1.0	(...,L,...; Dir), (...,L,...; Ab)	
8	..., L, ...	Ac	2.0	-1.0	(...,L,...; Ac), (...,L,...; Dir), (...,L,...; Ab)	
9	..., L, ...	Esq	2.0	-1.0	(...,L,...; Dir)	X

O programa agente Asp_RS_Parcial não cometeu falhas nas quatro primeiras interações que manteve com Amb. Os quatro primeiros episódios produzidos por Asp_RS_Parcial pertencem aos conjuntos de episódios ideais $Ep_{ideais}{}^K$ gerados nas quatro interações (K = 1, ..., 4) de Agent* com Amb. No quinto $Ep(P^5, A^5)$ = Ep(..., L,...; Esq) e no sexto episódio $Ep(P^6, A^6)$ = Ep(..., L,...; Ab), Asp_RS_Parcial cometeu falhas. Os episódios ideais produzidos por Agent* são $Ep(P^5, A^5*)$ = Ep(...,L,...; Dir) e $Ep(P^6, A^6*)$ = Ep(...,L,...; Dir), ou seja, existe uma ação melhor que 'Esq' no quinto episódio e que 'Ab' no sexto episódio. Em ambos os episódios, a ação 'Dir' é considerada melhor por levar o agente para uma sala vizinha suja.

A falha encontrada por ProMon nos episódios 5 e 6 sugere que quem está causando a falha é a função **ver**, visto que a função **ação** do agente poderia ser capaz de escolher a ação 'Dir' se soubesse que a sala à sua direita estava suja. Os episódios

produzidos por Asp_RS_Parcial nas interações 7 e 8 pertencem aos conjuntos de episódios ideais nestas interações, o agente testado movimentou-se para uma sala vizinha não visitada próxima a uma sala que contém sujeira.

Na interação 9, percebe-se que o episódio de Asp_RS_Parcial é diferente do episódio ideal produzido por Agent*, ou seja: $Ep(P^9, A^9) = Ep(...L..., Esq) \neq Ep((P^9, A^9*)) = Ep(...L..., Dir)$. Semelhantemente ao que aconteceu no quinto e no sexto episódio, o agente deixou de se movimentar para uma sala vizinha suja, assim, ProMon identificou uma falha. Além disso, Asp_RS_Parcial movimentou-se para uma sala anteriormente visitada e que estava limpa. Este tipo de falha no agente aspirador de pó Asp_RS_Parcial, pode-se considerar que a falha na função ver do agente pode ser controlada com a introdução de um módulo de atualização de estado interno, que está ausente no agente reativo simples com observabilidade parcial. A Tabela 5 ilustra o diagnóstico identificado por ProMon para uma história parcial de Asp_REiParcial.

Tabela 5. História parcial de Asp_REi_Parcial em Amb

K	Ep^K				$Ep_{ideais}{}^K$	Ep_{falha}
	P^K	A^K	$-av_E(P^K, A^K)$	$-av_L(P^K, A^K)$		
1	..., L, ...	Dir	1.0	-2.0	(...,L,...; Dir)	
2	..., L, ...	Ab	1.0	-2.0	(...,L,...; Ab)	
3	..., L, ...	Dir	1.0	-2.0	(...,L,...; Ab), (...,L,...; Dir)	
17	..., L,...	Esq	1.0	-2.0	(...,L,...; Esq)	
18	..., L,...	Esq	1.0	-2.0	(...,L,...; Dir)	X
19	..., L,...	Ac	1.0	-2.0	(...,L,...; Dir)	X

O programa agente Asp_REi_Parcial não cometeu falhas nas três primeiras interações, ou seja, os três primeiros episódios pertencem ao conjunto de episódios ideais produzidos por Agent*. O agente cometeu uma falha no episódio 18, ou seja, existe uma ação melhor que 'Esq', isto é, 'Dir' que leva o agente para uma sala vizinha suja. Semelhante ao Ep^{18}, o episódio 19 é diferente do episódio ideal produzido por Agent*, ou seja: $Ep(P^{19}, A^{19}) = Ep(\ldots L\ldots, Ac) \neq Ep(P^{19}, A^{19}*) = Ep(\ldots L\ldots, Dir)$. Em ambos os episódios, quem está causando a falha é a função **ver** do agente, visto que a função **ação** do agente poderia ser capaz de escolher a ação 'Dir' se soubesse que a sala à direita estava suja e, apesar dos valores de avaliação de Asp_REi_Parcial e Agent* serem iguais, o agente evita ganhar pontos ao não se movimentar para uma sala suja.

Assim, o agente aspirador de pó reativo simples com observabilidade parcial possui o pior desempenho na avaliação. Ao analisar as regras condição-ação do agente se confirmará que, por limitação da arquitetura, não são consideradas condições em seus antecedentes relacionadas a energia e a limpeza. Como o aspirador foi concebido como um agente reativo simples, pouco pode ser feito para melhorar seu desempenho, sendo necessário realizar uma extensão em sua estrutura para ampliar a observabilidade do ambiente ou acomodar um estado interno, o que permitirá ao agente economizar energia ao perceber em seu histórico de percepções que uma determinada sala já foi visitada.

6. Considerações Finais

Considerando que o agente racional deve ser capaz de realizar seus objetivos, testes adequados devem ser desenvolvidos para avaliar as ações e os planos executados pelo agente na realização destes objetivos. Para a realização dos testes em agentes racionais é necessário que técnicas que tratem da natureza peculiar do agente sejam aplicadas.

A abordagem apresentada considera que no caso dos agentes racionais, em que a medida de avaliação de desempenho é estabelecida pelo projetista, envolve vários objetivos que podem ser conflitantes. Os resultados obtidos demonstram que a abordagem proposta é eficaz, pois consegue gerar informações relevantes que permitem ao projetista raciocinar sobre o desempenho insatisfatório dos componentes na estrutura interna do programa do agente em teste. Porém ainda é necessário que se realize um estudo de caso com os agentes baseados em objetivos e em utilidade.

O aperfeiçoamento e a continuidade deste trabalho constituem-se oportunidade de trabalhos futuros que incluem: avaliação de diferentes ambientes de tarefa, utilização de outras técnicas de inteligência computação para selecionar casos de teste, investigação de outros aspectos que possam ser incluídos na medida de avaliação.

Referências

Holland, J. (1975) Adaptation in Natural and Artificial Systems. University of Michigan Press.

Houhamdi, Z. (2011) Test Suite Generation Process for Agent Testing. In: Indian Journal of Computer Science and Engineering (IJCSE), v. 2, n. 2.

Mylopoulos J.; Castro J. (2000) Tropos: A Framework for Requirements-Driven Software Development. Information Systems Engineering: State of the Art and Research Themes, Lecture Notes in Computer Science, Springer.

Nguyen, C. D.; Perini, A.; Tonella, P.; Miles, S.; Harman, M.; Luck, M. (2012) Evoluctionary Testing of Autonomous Software Agents. Autonomous Agents and Multi-Agent Systems. v. 25, n. 2, p. 260-283.

Nguyen, C. D. (2008) Testing Techniques for Software Agents. PhD Dissertation. University of Trento.

Pressman, R. S.; Maxim, B. (2014) Software Engineering: A Practitioner's Approach. 8 ed. McGraw-Hill.

Rocha, A. R. C.; Maldonado, J. C.; Weber, K. C. (2001) Qualidade de software – Teoria e prática. São Paulo: Prentice Hall.

Russell, S.; Norvig, P. (2013) Inteligência Artificial: uma abordagem moderna. 3 ed. São Paulo: Campus.

Silveira, F. R. V.; Campus, G. A. L.; Cortés, M. I. (2013) Rational Agents for the Test of Rational Agents. IEEE Latin America Transaction, v. 11, n. 1, feb.

Silveira, F. R. V.; Campos, G. A. L.; Cortés, M. I. (2014) Problem-Solving Agent to Test Rational Agents: A Case Study with Reactive Agents. In: 16th International Conference on Enterprise Information Systems (ICEIS). Lisboa, Portugal.

Sommerville, I. (2011) Engenharia de Software. 9 ed. São Paulo: Pearson Addison Wesley.

Wooldridge, M. (2002) An Introduction to MultiAgent Systems. John Wiley & Sons.

Zina, H. (2011) Test Suite Generation Process for Agent Testing. Indian Journal of Computer Science and Engineering (IJCSE), v. 2, n. 2.