# ProMon: agente de diagnóstico e monitoramento de falhas para agentes racionais

Janaide Nogueira de Sousa Ximenes [1]
Gustavo Augusto Lima de Campos [1]
Francisca Raquel de Vasconcelos Silveira [2]

**Resumo:** Tendo em vista a literatura acerca de testes de agentes, pode-se evidenciar uma lacuna em termos de técnicas efetivas utilizadas especificamente para testar agentes inteligentes. Este trabalho propõe uma abordagem para o teste de programas de agentes racionais. A abordagem consiste em um agente que monitora e realiza o diagnóstico de falhas no agente testado, identificando o subsistema que está gerando erros. Os experimentos buscaram avaliar a abordagem com programas de agentes reativos baseados em modelos e regras condição-ação, para resolver problemas em ambientes de tarefa parcialmente observáveis.

***Palavras-chave*:** Diagnóstico de Falhas, Monitoramento de Teste, Teste de Agentes Racionais.

**Abstract:** *In view of the literature on agents tests can show an effective gap in the specific techniques used for testing intelligent agents. This paper proposes an approach to the test of rational agents programs. The approach consists of an agent that monitors and performs the fault diagnosis in the test agent, identifying the subsystem that is generating errors. The experiments sought to evaluate the approach with reactive agents programs based on models and rules condition-action, to solve problems in partially observable task environments.*

***Keywords*:** *Diagnosis of Faults, Test Monitoring., Test Rational Agents,*

## 1 Introdução

A concepção de agentes inteligentes foi inicialmente introduzida na década de 70 [1]. Por serem bastante flexíveis, os agentes inteligentes podem ser aplicados nos mais diversos tipos de problemas, podem executar diferentes funções e modificar seu comportamento dinamicamente. Agentes são capazes de resolver problemas complexos, principalmente aqueles que os sistemas tradicionais não conseguem resolver [2].

Enquanto os agentes estão cada vez mais comuns, existe uma necessidade imprescindível para orientações das particularidades do processo de teste de agentes [3]. Entretanto tem sido escasso os estudos que visam propor métodos e técnicas eficazes e que garantam a qualidade dos sistemas baseados em agentes [4].

Uma das possíveis razões para a ausência de técnicas para um agente de teste é a dificuldade de aplicar as técnicas que são capazes de garantir a confiabilidade destes sistemas, devido as propriedades peculiares e a natureza específica de agentes de software, que se destinam a serem distribuídos e autônomos. Como consequência, a procura de um erro específico pode ser difícil, uma vez que você não pode reproduzi-lo a cada nova execução [5].

O agente percebe seu ambiente por intermédio dos sensores e age sobre o ambiente que o mesmo está inserido por meio dos atuadores. Sendo assim o agente verifica uma sequência de percepções(histórico completo

[1]Programa de Pós-Graduação em Ciências da Computação, Universidade Estadual do Ceará (UECE), *campus* Central, Av. Dr. Silas Munguba, 1700 - Fortaleza, CE, Brasil
{nogueirajanaide@gmail.com; gustavo@larces.uece.br}
[2]Instituto Federal do Ceará (IFCE), Rua Teófilo Pinto - Aracati, CE, Brasil
{raquel.vsilveira@hotmail.com}

**http://dx.doi.org/10.5335/rbca.v9i1.6533**

de todas as percepções do agente) e escolhe qual será a próxima ação [6]. O comportamento do agente é induzido pela associação entre o domínio da aplicação externa e um modelo interno, que constitui-se de uma base de conhecimento e um motor de inferência [7].

Para cada sequência de percepções possível, um agente racional escolhe uma ação visando alcançar o melhor resultado, se houver incertezas, o melhor resultado esperado conforme a medida de avaliação, que se refere a uma medida estabelecida para avaliar qualquer sequência dada dos estados do ambiente [6].

Para o teste de agentes, são considerados quatro tipos de testes: testes de agente único, teste de integração, teste de sistema e testes de aceitação [8]. O teste de agente único consiste em testar um único agente, com o objetivo testar a sua funcionalidade interior e capacidade do agente para cumprir os objetivos, de observar e executar ações no ambiente. No teste de integração a finalidade é se certificar de que um grupo de agentes e o ambiente trabalham corretamente em conjunto. O teste de sistema envolve a garantia de que todos os agentes do sistema trabalham em conjunto como pretendido. O teste de aceitação testa apenas sistemas multiagentes no ambiente de execução do cliente, com o intuito de verificar se ele se encontra com as características das partes interessadas[9].

Os testes podem ser classificados como: testes de caixa preta e teste de caixa branca. Os testes de caixa preta visam identificar as funcionalidades e a aderência aos requisitos, em uma visão externa ou do usuário, sem se basear em qualquer conhecimento da lógica interna e do código do componente a ser testado, já os testes de caixa branca avaliam as cláusulas de código, as configurações, a lógica interna do componente e outros elementos técnicos que possam ser relevantes[10]. Para esta abordagem estão contidos, testes caixa preta, aplicados no agente Thestes, e caixa branca, executados no agente ProMon.

O agente Thestes consiste em um agente para resolução de problemas de seleção de casos de teste que realiza uma busca local no espaço de estados de casos de teste orientado por utilidade. O agente ProMon é um agente reativo baseado em modelos que realiza o diagnóstico e o monitoramento de falhas no teste de agentes racionais. O agente Thestes recebe do Projetista um agente a ser testado, as medidas de avaliação que são necessárias e outras informações que podem ser necessárias para a realização do teste, como por exemplo, parâmetros de busca. O agente inicia a seleção dos casos de teste, testando Agent, agente a ser testado, em Amb, ambiente onde Agent será inserido.

O Thestes envia para o agente ProMon casos de teste em que Agent possui um comportamento mais inadequado, um conjunto de histórias correspondentes de Agent em Amb e os valores de avaliação de desempenho associados, episódio por episódio. O agente ProMon recebe essas informações e identifica para o Projetista os episódios em todas as histórias em que Agent falhou e os episódios ideais correspondentes, e uma identificação do tipo de falha, indicando o subsistema de Agent que ele presume esteja causando a falha.

A motivação deste trabalho está relacionada a ausência em termos de técnicas de ensaio aplicadas especificamente a agentes autônomos, de modo que se possa avaliar o comportamento e a confiança dos programas agentes. Este trabalho está dividido em cinco seções, onde a seção a seguir apresenta os trabalhos relacionados sobre a abordagem tratada, a seguinte aborda os tipos de agentes testados, apresenta o agente utilizado para a seleção dos casos de testes, como também o agente de monitoramento e diagnóstico de falhas. Logo após são evidenciados os resultados dos experimentos realizados para a abordagem, sequentemente são apresentadas as conclusões da abordagem.

## 2 Trabalhos relacionados

Esta seção apresenta uma revisão bibliográfica sobre algumas abordagens presentes na literatura para realizar os testes em agentes. É apresentado um resumo das abordagens descritas com suas principais características e a diferenciação entre as abordagens citadas e a abordagem proposta.

Nesta seção, são considerados os seguintes critérios, com o propósito de avaliar o desempenho das abordagens existentes para o teste de programas de agente: (i) a utilização da noção de agentes racionais, (ii) a utilização de casos de teste gerados de acordo com os objetivos do agente, (iii) o emprego de uma medida para avaliar o desempenho do agente, (iv) avaliação dos planos usados pelo agente para alcançar os objetivos, (v) consideração dos planos que são necessários para que este agente alcance estes objetivos, e (vi) monitoramento e diagnóstico da ação do agente testado e o objetivo do agente.

Lam e Barber [11], propõem um processo para compreender os comportamentos do agente de software. A abordagem imita o que um usuário humano testador faz na compreensão software, que são: construir e aperfeiçoar uma base de conhecimento sobre os comportamentos dos agentes, e usá-la para verificar e explicar os comportamentos dos agentes em tempo de execução. Com base na análise dos critérios para avaliação das abordagens, a abordagem não dispõe da utilização da noção de agentes racionais, não lida com a geração de casos de teste, não aborda o monitoramento e diagnóstico da ação do agente testado em nível de teste caixa branca.

Houhamdi [3], apresenta uma abordagem de teste orientada a objetivos para programas de agentes. Na abordagem é especificado um processo de teste que complementa a metodologia Tropos [12] e reforça a relação mútua entre a análise de objetivos e testes. Na abordagem é proposto a derivação de casos de teste a partir da análise de artefatos de requisitos orientado aos objetivos do agente para a realização dois níveis de teste: unitário, para certificar de que todas as unidades que fazem parte de um agente, tais como metas, planos, base de conhecimento, funcionam conforme projetado, e agente, que corresponde a testar a integração dos diferentes módulos dentro de um agente, pode-se perceber que a abordagem utiliza os testes de caixa branca e caixa preta.

A metodologia de desenvolvimento Tropos[12], relaciona os objetivos aos casos de teste e realiza um processo sistemático para derivar casos de teste, a partir da análise dos objetivos.

A metodologia proposta contribui para as metodologias Agent-Oriented Software Engineering (AOSE). Apesar de utilizar a análise de objetivos, essa estratégia não apresenta a noção de agentes racionais de Russell e Norvig [6], não há uma medida de avaliação de desempenho e uma simulação que possa monitorar o comportamento do agente ao executar as ações que envolvem os objetivos do agente.

Nguyen et. al [5], propõem uma metodologia que consiste em: representar os objetivos como função de qualidade( os objetivos que são necessários para avaliar o agente são transformados em uma função de qualidade para mensurar a satisfação dos stakeholders). O processo de geração de casos de teste é realizado a partir dos testes evolucionários, em que é gerado a população inicial, logo após, é realizado a execução do teste e o monitoramento, em seguida, é realizada a coleta dos dados e a reprodução. Nesta abordagem não há uma simulação que possa monitorar e diagnosticar o comportamento do agente ao executar uma ação que envolve os objetivos ou metas.

## 3 Abordagem

### 3.1 Tipos de agentes

Este trabalho leva em consideração os quatro tipos básicos de agentes racionais: reativos simples, reativos baseado em modelo, baseados em objetivos e baseados em utilidades [6]. Entretanto a avaliação desta abordagem considera apenas os agentes reativos simples e reativos baseados em modelos. O agente reativo simples não considera o histórico de percepções, ou seja, seleciona a ação a partir da percepção atual do ambiente. Comportamentos reativos simples acontecem mesmo em ambientes mais complexos, como é o caso de um motorista de táxi automatizado [6]. O agente reativo baseado em modelos possui uma estrutura apropriada às situações em que o ambiente de execução, nos quais estão introduzidos é observável, entretanto a geografia desse ambiente não é conhecida [6].

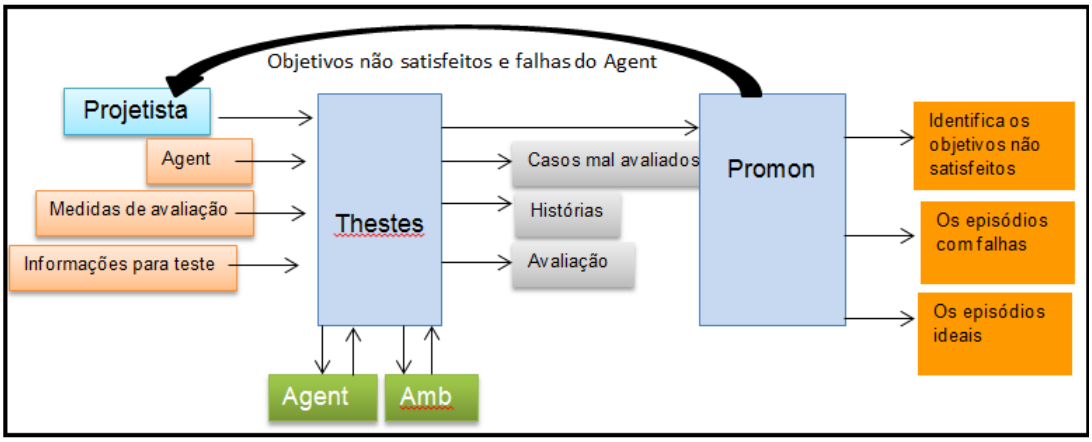
A abordagem proposta para o teste de agentes racionais fundamenta-se em [13] que considera a noção de agentes racionais na seleção de casos de teste. Os casos de teste são gerados de acordo com (i) os objetivos contidos na medida de avaliação de desempenho do agente testado, (ii) a simulação das interações do agente testado com seu ambiente (histórias) e (iii) estratégias de busca local multiobjetivos orientadas por utilidade para encontrar os casos de teste e as histórias correspondentes em que o agente não foi bem avaliado.

De acordo com a Figura 1, pode-se analisar que a abordagem considera um Projetista interagindo com programas de agentes envolvidos, ou seja, o programa agente a ser testado, Agent, concebido pelo Projetista, o programa de ambiente de tarefa, Amb, um programa agente para a seleção de casos de teste, Thestes, e um agente para monitorar e diagnosticar falhas em Agent, ProMon.

### 3.2 Agente para seleção de casos de teste

O agente Thestes consiste em um agente de resolução de problemas de seleção de casos de testes que possui a estrutura do programa agente orientado por utilidade e realizando busca local, baseada em populações e orientada

Figura 1: Estrutura geral da abordagem




Fonte: Adaptado de Silveira[13].

por uma função utilidade, para encontrar conjuntos de casos de teste satisfatórios.

O agente seleciona os casos de teste e envia para o agente ProMon, um conjunto de soluções eficientes determinada pela estratégia de busca multiobjetivo, ou seja, descreve os casos de teste em que o agente a ser testado possui o comportamento mais inadequado, um conjunto de histórias correspondentes e seus valores de utilidade. O problema de seleção de casos de teste foi formulado como um problema multiobjetivo e a função utilidade representa o negativo da função avaliação de desempenho especificada pelo Projetista.

O agente Thestes possui três subsistemas. No subsistema de percepção (ver) são mapeadas as informações essenciais ao teste de Agent, em uma representação computacional, $Estado_k$, adequada ao processamento dos outros dois subsistemas (próximo e ação): (*Agent*, *Amb*, *Parâmetros*BUSCA, *ParâmetrosSimulação*). O subsistema de atualização de estado interno, próximo, armazena as informações em $Estado_k$, considera os parâmetros em *ParâmetrosSimulação* e *GeradorCasos*TEST e gera um conjunto inicial *Casos*TEST de maneira aleatória ou, dependendo dos parâmetros, gera casos específicos, conforme o Projetista desejar testar certos aspectos da estrutura interna de *Agent* e, consequentemente, facilitar a próxima etapa de tomada de decisão (ação): (*Casos*TEST, *Agent*, *Amb*, *Parâmetros*BUSCA, *ParâmetrosSimulação*).

A função ação de Thestes inicia um processo de busca local visando encontrar uma ação satisfatória$Ao_k$. Esta função utiliza informações a respeito de um modelo de transição de estados, *ModeloTransição*, para gerar novos casos de teste a partir de *Casos*TEST, e o protocolo de interação, *ProtocoloInteração*, e uma função utilidade, *Utilidade*, para, respectivamente, obter as histórias correspondentes aos casos de teste no conjunto e avaliar o desempenho de Agent nestas histórias. Assim, enquanto o conjunto *Casos*TEST inicial e o modelo de transição permitem que ação gere o espaço de estados do problema de seleção, o protocolo e a função utilidade permitem que ação avalie os estados gerados (conjuntos de histórias) e, consequentemente, o comportamento de Agent nestes estados.

As informações geradas pela função ação, Thestes envia para o agente ProMon três informações importantes em $Ao_k$: (i) o conjunto *Casos*TEST corrente como solução para o problema e os melhores casos encontrados até a solução, (ii) as histórias correspondentes de Agent em Amb contidas no conjunto *Histórias*, (iii) os valores de desempenho, considerando cada um dos objetivos na medida de avaliação.

### 3.3 Abordagem ProMon

O agente ProMon é um agente reativo baseado em modelos que recebe informações do agente Thestes, que são, um conjunto de soluções eficientes, determinadas pela estratégia de busca multiobjetivo que pode ser compreendida como um, conjunto de histórias correspondentes as soluções e seus valores de utilidade e identifica para o Projetista: (i)os objetivos na medida de avaliação que não estão sendo satisfeitos adequadamente pelo Agent,

(ii) os episódios nas histórias de Agent em Amb que são falhas, ou seja, que estão "distantes" do ideal, (iii) quais são os episódios ideais correspondentes, e (iv) os subsistemas que possivelmente estão gerando falhas.

O agente ProMon considera que os agentes testados foram concebidos levando em consideração a descrição de arquitetura abstrata de agentes proposta em [14]. Esta arquitetura pode ser concretizada de diversas formas diferentes, entretanto todas as formas podem ser visualizadas a princípio como decompostas em três subsistemas principais. O subsistema de percepção executa o mapeamento de uma informação sobre a percepção, **ver:** P $\rightarrow$ Estado. O subsistema de atualização de estado interno atualiza o estado interno, mapeando a representação da percepção atual e a informação sobre o estado interno mantido pelo agente em um novo estado interno, **próximo:** Estado x EI $\rightarrow$ EI. E o subsistema de tomada de decisão mapeia uma informação sobre o estado interno do agente em uma ação correspondente, **ação:** EI $\rightarrow$ A.

O agente de monitoramento e diagnóstico de falha, foi proposto em duas etapas, correspondentes ao processamento de um subsistema do tipo próximo e o segundo do tipo ação. O primeiro subsistema, a função próximo do agente recebe as histórias associadas aos casos em CasosTEST H(CasosTEST) e identifica todos os episódios que possuam falhas nessas histórias.

Dessa maneira, como no caso do agente Thestes, esta função considera o programa ambiente Amb, o protocolo ProtocoloInteração e uma versão totalmente observável do agente testado (onipresente), denotada por Agent* para detectar se um episódio em uma interação K, em uma história associada a um caso i em CasosTEST, e gerar dois conjuntos de episódios: o conjunto de episódios ideais na interação, ${Ep_{ideais}}^K$ e o conjunto de episódios com falhas nas histórias, $Ep_{falhas}$. Seja:

- **h(**$Caso_i$**)** $\in (PxA)^{NInt}$ história de comprimento $N_{Int}$ de Agent em Amb equivalente ao $Caso_i \in$ CasosTEST;
- $h^*(Caso_i) \in (PxA)^{NInt}$ uma história de comprimento $N_{Int}$ de Agent* em Amb que equivale ao $Caso_i \in$ CasosTEST
- $Ep^K$ (h($Caso_i$)) $\in$ PxA ou, Ep(($P^K$, $A^K$)) $\in$ PxA um episódio na interação k da história de Agent em Amb correspondente ao caso $Caso_i \in$ CasosTEST.
- $Ep^K$(h*($Caso_i$)) $\in$ PxA ou, mais especificamente, Ep(($P^K$, $A^{K*}$))) $\in$ PxA um episódio na interação K da história de Agent* em Amb correspondente ao caso $Caso_i \in$ CasosTEST.

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

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

$$av_m(Ep(P^K, A^{K*})) \geq av_m(Ep((P^K, A^K)));$$

- (2) $A^{K*}$ é equivalente ou melhor a $A^K$ levando em consideração a perspectiva do projetista.

Os episódios $Ep(P^K, A^K)$, que são gerados pelo Agent que não pertencem ao conjunto ${Ep_{ideais}}^K$, devem constituir o conjunto de episódios com falha. A segunda condição depende da opinião do projetista. Essa visa identificar falhas que não são percebidas diretamente na especificação da medida de avaliação de desempenho.

A Figura 2 ilustra uma medida de avaliação de desempenho para o caso de um agente aspirador de pó que deve maximizar os níveis de limpeza do ambiente e de energia em sua bateria e limpar um ambiente. A segunda coluna representa uma parte das informações nas percepções do agente em cada episódio. A terceira coluna descreve as ações possíveis nesses episódios Direita (Dir), Esquerda (Esq), (Aspirar (Asp), Acima (Ac), Abaixo (Ab) e Não operar (N-op)). Na quarta coluna está associada ao objetivo de energia e a quinta coluna está relacionada aos objetivos de limpeza, as duas funções escalares ($av_E$ e $av_L$) possuem o objetivo de medir o desempenho do agente em cada episódio de sua história no ambiente.

Considerando a medida de avaliação de desempenho é possível perceber que as linhas 1, 5 e 6 detectam episódios que são falhas, isto é, o agente aspirar em uma sala que já está limpa, e movimentar-se para uma sala

Figura 2: Medida de avaliação de desempenho

| Linhas | $Ep^k$=(Percepção$^k$, | Ação$^k$) | avE($Ep^k$) | AvL($Ep^k$) |
|---|---|---|---|---|
| 1 | ..., L, ... | Asp | -1.0 | 0.0 |
| 2 | ..., L, ... | Dir, Esq, Ac, Ab | -2.0 | 1.0 |
| 3 | ..., L, ... | N-op | 0.0 | 0.0 |
| 4 | ..., S, ... | Asp | -1.0 | 2.0 |
| 5 | ..., S, ... | Dir, Esq, Ac, Ab | -2.0 | -1.0 |
| 6 | ..., S, ... | N-op | 0.0 | -1.0 |
| ... | ... | ... | ... | ... |

Fonte: Adaptado de Silveira, 2013[13].

vizinha ou não operar quando a sala corrente está suja. Porém, não são identificados 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. Para solucionar o problema, a condição (2) deverá ser especificada de acordo com o domínio de aplicação do agente testado Agent, solucionando assim o problema.

Encerrando a etapa de identificação dos episódios que são falhas em todas as interações k nas histórias associadas aos casos em CasosTEST, realizada pela função próximo do agente ProMon, a segunda etapa do processo de monitoramento e diagnóstico do agente pode ser iniciada. Nesta etapa a função ação do agente utiliza o conjunto de episódios $Ep_{falhas}$ e uma função ação baseada em regras condição-ação para identificar o tipo de falha associada ao episódio com falha em uma interação específica Ep(($P^K$, $A^K$)), considerando o próprio episódio, os valores de avaliação associados ao episódio em todos os atributos m na medida de avaliação $av_m$(Ep(($P^K$, $A^K$))), e os episódios ideais Ep(($P^K$, $A^{K*}$)) em ${Ep_{ideais}}^K$, produzidos por Agent* na mesma interação.

A Figura 3 apresenta as regras genéricas utilizadas pelo agente de monitoramento e diagnóstico para situações em que Agent é um agente reativo simples. Já a Figura 4 demonstra as regras genéricas utilizadas pelo agente ProMon para situações em que Agent é um agente reativo baseado em modelos.

Figura 3: Regras do agente ProMon para agentes reativos simples

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

Fonte: Silveira, 2013[13].

Os antecedentes nas regras das figuras, Figura 3 e Figura 4, consistem em duas condições associadas às condições (1) e (2) descritas anteriormente. Estas condições associadas indicam a razão pela qual um episódio com falhas Ep(($P^K$,$A^K$)) foi inserido no conjunto $Ep_{falhas}$, isto é:

- (1)' existe no mínimo um atributo z em que $av_z$(Ep(($P^K$, $A^{K*}$))) > $av_z$(Ep(($P^K$, $A^K$))), enquanto que no

restante dos atributos $av_m$(Ep(($P^K$, $A^{K*}$))) $\geq$ $av_m$(Ep(($P^K$, $A^K$)));

- (2)' $A^{K*}$ é melhor que $A^K$.

Os consequentes nas regras refere-se as mensagens enviadas ao projetista sobre Agent. A primeira mensagem indica a condição que foi atendida, que é a razão para o episódio pertencer ao conjunto $Ep_falha$. A segunda mensagem é uma disjunção envolvendo regras no formato "consequente se antecedente", ou seja, "Falha no Módulo X **se** Condição Y for satisfeita". Estas mensagens são enviadas ao Projetista.

As regras são disponibilizadas ao projetista para que o mesmo avalie as condições expostas em seus antecedentes e perceba quais módulos de processamento de informação em Agent que possivelmente estão causando as falhas, conforme as condições (1)' e (2)'.

Deste modo, como o antecedente em cada uma das regras propostas é uma conjunção de proposições envolvendo as saídas dos módulos de processamento da informação de Agent e do agente com observabilidade total Agent*, a abordagem com o agente ProMon pressupõe que o projetista tem o controle do agente testado e é capaz de comparar o processamento realizado pelos módulos de Agent com o processamento realizado pelos módulos de Agent*. As regras da Figura 3 são aplicadas aos casos em que Agent é um agente reativo simples. Já as regras da Figura 4, são utilizadas quando Agent for um programa agente reativo baseado em modelos.

A primeira e segunda regras indicam que a falha está no subsistema de percepção de Agent, quando o subsistema de percepção de Agent* gera uma informação $Estado^K$ que difere da gerada pelo subsistema de Agent e o projetista presume que o subsistema de tomada de decisão de Agent estaria apto para selecionar uma ação ideal $A^{K*}$, isto é, uma das ações selecionadas por Agent* e existente em ${Ep_{ideais}}^K$, caso possuísse a informação $Estado^K$. Caso não exista falha em ver, as regras indicam que a falha está no subsistema de tomada de decisão de Agent, ou seja, apesar de perceber como Agent*, o projetista supõe que o agente testado Agent não conseguiria tomar decisões equivalentes. Além das duas possibilidades, as regras indicam também que falha pode estar presente nos dois subsistemas.

Figura 4: Regras do agente ProMon para agentes reativos baseados em modelos

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

Fonte: Silveira, 2013[13].

Idêntico, para o caso em que Agent dispõe de um estado interno, a terceira e quarta regras indicam que a falha está na função próximo quando o estado interno de Agent for distinto da informação produzida pelo subsistema de percepção de Agent* e o projetista supor que o subsistema de tomada de decisão de Agent estaria apto para selecionar uma ação ideal $A^{K*}$ para o $Estado^K$. Considerando que não existe a falha em próximo, essas regras indicam que a falha está no subsistema de tomada de decisão do agente reativo baseado em modelos Agent. Por fim, as regras indicam que a falha pode estar presente tanto em próximo como em ação.

É interessante evidenciar que as regras aplicadas ao agente reativo baseado em modelos não consideram

a possibilidade de falha na função ver deste agente, ou seja, especificamente neste caso, o projetista já conhece as limitações do agente em termos de observabilidade do ambiente e, por isso, concebeu uma função próximo, visando minimizar o baixo nível de confiabilidade da função ver. Portanto, para o projetista, é mais importante perceber se existem falhas na função próximo.

Embora não especificado, as regras adotadas para este agente podem ser adaptadas para outros tipos de agentes, já que estes agentes também podem ser descritos em termos dos componentes ver, próximo e ação.

Para este trabalho foram avaliados dois tipos de programas agentes aspirador de pó, os reativos simples e os reativos baseados em modelos com observabilidade parcial.

## 4 Experimentos

Esta seção apresenta as regras dos agentes testados e os testes realizados para avaliar os resultados produzidos pela abordagem, testando os agentes aspiradores de pó, considerando a monitorização e diagnóstico de falhas do agente aspirador realizado pelo agente ProMon. Foram testados agentes aspirador de pó reativo baseado em modelos em que o ambiente é parcialmente observável e agentes reativos simples com observabilidade parcial.

O programa agente aspirador de pó reativo baseado em modelos em que o ambiente é parcialmente observável(RM_Parcial) contém um estado interno que guarda o histórico das percepções obtidas e uma ação é escolhida levando em consideração esta informação. A Figura 5 apresenta as regras condição-ação do agente RM_Parcial.

Figura 5: Regras condição-ação de agente RM_Parcial

**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

O programa agente aspirador de pó reativo simples (RS_Parcial) enfatiza a seleção das ações com base na percepção atual, ignorando o histórico de percepções obtido, em um ambiente parcialmente observável, isto é, a função ver de RS_Parcial possibilita perceber apenas o estado, sujo ou limpo, da sala em que o agente está. A Figura 6 apresenta as regras condição-ação de RS_Parcial.

Figura 6: Regras condição-ação de agente RS_Parcial

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

A tarefa de manter um ambiente limpo é aparentemente simples, porém para manter o ambiente limpo e sem desperdiçar energia é necessário a utilização de módulos de operação e tomada de decisão. Um programa agente aspirador de pó deve limpar um ambiente e maximizar os níveis de limpeza do ambiente e de energia em sua bateria ao final da tarefa [13]. A Figura 7 indica o significado de cada símbolo empregado na representação dos estados do ambiente.

Para a realização dos experimentos foram consideradas as situações em que o ambiente que interage com o agente a ser testado é um ambiente parcialmente observável, ou seja, o agente possui uma percepção limitada do ambiente. Pressupõe-se que o aspirador possui sensores que percebem o ambiente, mas que a sua função ver consegue mapear apenas uma pequena representação da informação, isto é, o estado da sala em que o agente se localiza.

Figura 7: Significado dos símbolos

| Símbolo | Significado |
|---|---|
| 0 | sala limpa |
| 1 | sala suja |
| **0** | sala limpa visitada |
| **1** | sala suja visitada |
| **0** | agente está em sala limpa |
| **1** | agente está em sala suja |
| ***0*** | agente estava em sala limpa |
| ***1*** | agente estava em sala suja |
| **0** | agente está em sala limpa visitada |
| **1** | agente está em sala suja visitada |
| ***0***, | agente está em sala limpa e última ação foi na sala |
| ***1*** | agente está em sala suja e última ação foi na sala |

O programa agente RM_Parcial(reativo baseado em modelos com observabilidade parcial) não cometeu falhas nas primeiras interações que manteve com Amb. A Figura 8 ilustra os episódios 17 ao 22 da história de Agent em Amb em que RM_Parcial cometeu falhas nos episódios Ep18, Ep19 e Ep20.

Figura 8: Episódios na história de RM_Parcial

| **História Realizada – $Ep^{17}$, $Ep^{18}$, $Ep^{19}$, $Ep^{20}$, $Ep^{21}$, $Ep^{22}$** | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **-av$_E$** | **-av$_L$** | **-av$_E$** | **-av$_L$** | **-av$_E$** | **-av$_L$** | **-av$_E$** | **-av$_L$** | **-av$_E$** | **-av$_L$** | **-av$_E$** | **-av$_L$** |
| 1.0 | -2.0 | 2.0 | -1.0 | 2.0 | -1.0 | 1.0 | -2.0 | 2.0 | -1.0 | 2.0 | -1.0 |
| **Falha** | | **Falha** | | **Falha** | | **Falha** | | **Falha** | | **Falha** | |
| Não | | **Sim** | | **Sim** | | **Sim** | | Não | | Não | |
| [**0**, **0**, 0, 0, 0] | | [**0**, **0**, 0, 0, 0] | | [**0**, **0**, 0, 0, 0] | | [**0**, **0**, 0, 0, 0] | | [**0**, **0**, 0, 0, 0] | | [**0**, **0**, 0, 0, 0] | |
| [0, **0**, **0**, **0**, **0**] | | [0, **0**, **0**, **0**, **0**] | | [0, **0**, **0**, **0**, **0**] | | [0, **0**, **0**, **0**, **0**] | | [0, **0**, **0**, **0**, **0**] | | [0, **0**, **0**, **0**, **0**] | |
| [0, 0, **0**, **0**, **0**] | | [0, 0, **0**, **0**, **0**] | | [0, 0, **0**, **0**, **0**] | | [**0**, 0, **0**, **0**, **0**] | | [***0***, **0**, **0**, **0**, **0**] | | [**0**, ***0***, **0**, **0**, **0**] | |
| [0, 0, **0**, 0, 0] | | [0, 0, **0**, 0, 0] | | [**0**, 0, **0**, 0, 0] | | [***0***, 0, **0**, 0, 0] | | [**0**, 0, **0**, 0, 0] | | [**0**, **0**, **0**, 0, 0] | |
| [0, ***0***, **0**, 0, 1] | | [**0**, ***0***, **0**, 0, 1] | | [***0***, **0**, **0**, 0, 1] | | [**0**, **0**, **0**, 0, 1] | | [**0**, **0**, **0**, 0, 1] | | [**0**, **0**, **0**, 0, 1] | |

O agente RM_Parcial cometeu uma falha no episódio realizado na Figura 8, ou seja, existe uma ação melhor que 'Esq' no $Ep^{18}$, isto é, 'Dir' que leva o agente para uma sala vizinha que contém sujeira. Neste caso, indica que quem está gerando a falha é a função ver do agente, dado que a função ação de RM_Parcial poderia ser capaz de escolher a ação 'Dir' se soubesse que a sala à sua direita estava suja. Neste caso, apesar dos valores de avaliação de RM_Parcial e Agent* serem iguais, o agente evita ganhar pontos ao deixar de se movimentar para uma sala suja.

A Figura 9, demonstra o episódio Ep($(P^{18}$, $A^{18})$) = EP((. . . L. . . , Esq)) de RM_Parcial em Amb e o episódio ideal Ep($(P^{18})$) = EP((. . . L. . . , Dir) produzido por Agent*. Cada coluna na Figura 9 corresponde a um conjunto de episódios ideais em uma interação.

Figura 9: $Ep^{18}$ na história de RM_Parcial

| Histórias – $Ep^{18}$ | | | |
|---|---|---|---|
| **Realizada** | | **Ideal** | |
| **-av$_E$** | **-av$_L$** | **-av$_E$** | **-av$_L$** |
| 2.0 | -1.0 | 2.0 | -1.0 |
| **Falha** | | | |
| **Sim** | | | |
| [**0**, **0**, 0, 0, 0] | | [**0**, **0**, 0, 0, 0] | |
| [0, **0**, **0**, **0**, **0**] | | [0, **0**, **0**, **0**, **0**] | |
| [0, 0, **0**, **0**, **0**] | | [0, 0, **0**, **0**, **0**] | |
| [0, 0, **0**, 0, 0] | | [0, 0, **0**, 0, 0] | |
| [**0**, ***0***, **0**, 0, 1] | | [0, ***0***, **0**, 0, 1] | |

Figura 10: $Ep^{19}$ na história de RM_Parcial

| Histórias – $Ep^{19}$ | | | |
|---|---|---|---|
| **Realizada** | | **Ideal** | |
| **-av$_E$** | **-av$_L$** | **-av$_E$** | **-av$_L$** |
| 2.0 | -1.0 | 2.0 | -1.0 |
| **Falha** | | | |
| **Sim** | | | |
| [**0**, **0**, 0, 0, 0] | | [**0**, **0**, 0, 0, 0] | |
| [0, **0**, **0**, **0**, **0**] | | [0, **0**, **0**, **0**, **0**] | |
| [0, 0, **0**, **0**, **0**] | | [0, 0, **0**, **0**, **0**] | |
| [**0**, 0, **0**, 0, 0] | | [0, 0, **0**, 0, 0] | |
| [***0***, **0**, **0**, 0, 1] | | [***0***, **0**, **0**, 0, 1] | |

Conforme análise da Figura 10, percebe-se que o $EP^{19}$ de RM_Parcial é diferente do episódio ideal produzido por $Agent^*$, ou seja: Ep(($P^{19}$, $A^{19}$)) = Ep((...L..., Ac) - $\neq$ Ep(($P^{19}$, $A^{19*}$)) = Ep((...L..., Dir)).

Semelhantemente ao que aconteceu com a interação 18, o agente, no $Ep^{19}$, deixou de se movimentar para uma sala vizinha que continha sujeira, assim, identifica-se que a função ver está gerando a falha.

A Figura 11 apresenta o episódio Ep(($P^{20}$, $A^{20}$)) = Ep((...L..., Ac)) e o conjunto de episódios ideais gerado por $Agent^*$ na interação 20 para RM_Parcial, Ep(($P^{20}$, $A^{20*}$)) = Ep((...L..., Dir), (...L..., Ab)).

O programa agente RS_Parcial (reativo simples com observabilidade parcial) não cometeu falhas nas quatro primeiras interações que manteve com o ambiente, ou seja, os quatro episódios produzidos por RS_Parcial pertencem ao conjunto de episódios ideais ${Ep_{ideais}}^K$ gerados nas quatro interações (K = 1, ..., 4) de $Agent^*$ com Amb.

A Figura 12 apresenta um episódio em que o agente RS_Parcial cometeu uma falha. Pode-se perceber que no quinto episódio Ep(($P^5$, $A^5$)) = Ep((...L..., Esq)) de RS_Parcial em Amb e o episódio ideal seria Ep(($P^5$, $A^{5*}$)) = Ep((...L..., Dir)) produzido por $Agent^*$.

O agente RS_Parcial cometeu uma falha no episódio realizado, ou seja, existe uma ação melhor que 'Esq' no quinto episódio, isto é, 'Dir' que leva Agent para uma sala vizinha que estava suja. Neste caso, indica que quem está causando a falha é a função ver do agente, visto que a função ação de RS_Parcial poderia ser capaz de

Figura 11: $Ep^{20}$ na história de RM_Parcial

| Histórias – $Ep^{20}$ | | | | | |
|---|---|---|---|---|---|
| **Realizada** | | **Ideal** | | | |
| **-av$_E$** | **-av$_L$** | **-av$_E$** | **-av$_L$** | **-av$_E$** | **-av$_L$** |
| 2.0 | -1.0 | 2.0 | -1.0 | 2.0 | -1.0 |
| **Falha** | | | | | |
| **Sim** | | | | | |
| [**0**, **0**, 0, 0, 0] | | [**0**, **0**, 0, 0, 0] | | [**0**, **0**, 0, 0, 0] | |
| [0, **0**, **0**, **0**, **0**] | | [0, **0**, **0**, **0**, **0**] | | [0, **0**, **0**, **0**, **0**] | |
| [**0**, 0, **0**, **0**, **0**] | | [0, 0, **0**, **0**, **0**] | | [0, 0, **0**, **0**, **0**] | |
| [***0***, 0, **0**, 0, 0] | | [***0***, 0, **0**, 0, 0] | | [***0***, 0, **0**, 0, 0] | |
| [**0**, **0**, **0**, 0, 1] | | [**0**, **0**, **0**, 0, 1] | | [**0**, **0**, **0**, 0, 1] | |

Figura 12: $Ep^{5}$ na história de RS_Parcial

| Histórias – $Ep^{5}$ | | | |
|---|---|---|---|
| **Realizada** | | **Ideal** | |
| **-av$_E$** | **-av$_L$** | **-av$_E$** | **-av$_L$** |
| 2.0 | -1.0 | 2.0 | -1.0 |
| **Falha** | | | |
| **Sim** | | | |
| [**0**, **0**, 0, 0, 1] | | [**0**, **0**, 0, 0, 1] | |
| [0, **0**, 1, 1, 0] | | [0, **0**, 1, 1, 0] | |
| [**0**, ***0***, 1, 1, 1] | | [0, ***0***, **1**, 1, 1] | |
| [0, 0, 0, 1, 1] | | [0, 0, 0, 1, 1] | |
| [0, 0, 1, 1, 1] | | [0, 0, 1, 1, 1] | |

escolher a ação 'Dir' se soubesse que a sala à sua direita estava suja. Neste caso, apesar dos valores de avaliação de RS_Parcial e $Agent^*$ possuírem valores iguais, Agent se abstém de ganhar mais pontos ao deixar de se movimentar para uma sala suja.

O programa agente RS_Parcial não cometeu falhas nos três episódios seguintes, estes pertencem ao conjunto de episódios ideais ${Ep_{ideais}}^K$ gerados nas interações (K = 6, 7 e 8). A Figura 13 ilustra o nono episódio gerado por RS_Parcial e por Agent*.

Pode-se perceber a partir da Figura 13 que o episódio de RS_Parcial é distinto de ${Ep_{ideais}}^K$ produzido por $Agent^*$, isto é: Ep(($P^9$, $A^9$)) = Ep((...L..., Esq)) $\neq$ Ep(($P^9$, $A^{9*}$)) = Ep((...L..., Dir)). Semelhantemente ao que aconteceu no $Ep^5$, o agente deixou de se movimentar para uma sala vizinha que estava suja. Porém, diferentemente do que ocorreu anteriormente, o RS_Parcial movimentou-se para uma sala que já havia sido visitada e que já estava limpa, portanto, identificou-se uma falha, sendo considerada na função ver do agente, e que pode ser controlada com a introdução de um módulo de atualização de estado interno, que está ausente no agente reativo simples com observabilidade parcial.

Figura 13: $Ep^9$ na história de RS_Parcial

| Histórias – $Ep^9$ | | | |
|---|---|---|---|
| Realizada | | Ideal | |
| $-av_E$ | $-av_L$ | $-av_E$ | $-av_L$ |
| 2.0 | -1.0 | 2.0 | -1.0 |
| Falha | | | |
| Sim | | | |
| [0, 0, 0, 0, 1] | | [0, 0, 0, 0, 1] | |
| [0, 0, 1, 1, 0] | | [0, 0, 1, 1, 0] | |
| [0, 0, 1, 1, 1] | | [0, 0, 1, 1, 1] | |
| [0, 0, 0, 1, 1] | | [0, 0, 0, 1, 1] | |

## 5 Conclusão

A abordagem tratada considera, no caso dos agentes racionais, uma medida de avaliação, em geral, envolve mais de um objetivo e estes podem ser conflitantes entre si. O resultado dos testes pode indicar o desempenho médio do agente e, principalmente, os objetivos que não estão sendo satisfeitos pelo agente. Além de informações sobre as interações entre o agente e seu ambiente que sejam úteis para a identificação das falhas, por exemplo, os episódios problemáticos e os comportamentos insatisfatórios do agente e realizar mudanças que são necessários para que o agente melhore seu desempenho.

Atrelando o contexto de Engenharia de Software e Inteligência Artificial, presumindo que uma boa avaliação do agente depende dos casos de teste selecionados e do diagnóstico do agente ProMon. Considera-se que os objetivos do trabalho foram satisfeitos ao longo de seu desenvolvimento. Já que o agente ProMon foi capaz de indicar falhas nos componentes, como no caso do agente deixar de se locomover para uma sala que contém sujeira, o agente ProMon identifica que a falha está na função ver. Ou até mesmo, quando o agente se locomove para uma sala que já foi visitada

Para trabalhos futuros sugere-se o desenvolvimento de testes em programas agentes baseados em utilidade e baseados em objetivos, bem como verificação de outros aspectos que podem ser incluídos no diagnóstico do agente ProMon e que contribuem para a identificação de episódios problemáticos.

## Referências


[1] HEWITT, C. Viewing control structures as patterns of passing messages. *Artificial intelligence*, Elsevier, v. 8, n. 3, p. 323–364, 1977.

[2] FRIGO, L. B.; POZZEBON, E.; BITTENCOURT, G. O papel dos agentes inteligentes nos sistemas tutores inteligentes. In: *World Congress on Engineering and Technology Education*. [S.l.: s.n.], 2004. p. 86.

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

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

[5] NGUYEN, C. D. et al. Evolutionary testing of autonomous software agents. *Autonomous Agents and Multi-Agent Systems*, Springer, v. 25, n. 2, p. 260–283, 2012.

[6] NORVIG, P.; RUSSELL, S. *Inteligência Artificial, 3ª Edição*. [S.l.]: Elsevier Brasil, 2014. v. 1.

[7] TECUCI, G. *Building intelligent agents: an apprenticeship multistrategy learning theory, methodology, tool and case studies*. [S.l.]: Morgan Kaufmann, 1998.

[8] NGUYEN, C. D.; PERINI, A.; TONELLA, P. Goal-oriented testing for mass. *International Journal of Agent-Oriented Software Engineering*, Inderscience Publishers, v. 4, n. 1, p. 79–109, 2009.

[9] ZINA, H. Test suite generation process for agent testing. *Indian Journal of Computer Science and Engineering (IJCSE)*, Citeseer, v. 2, n. 2, 2011.

[10] RIOS, E. *Teste de software*. [S.l.]: Alta Books Editora, 2006.

[11] LAM, D. N.; BARBER, K. S. Debugging agent behavior in an implemented agent system. In: *Programming Multi-Agent Systems*. [S.l.]: Springer, 2004. p. 104–125.

[12] MYLOPOULOS, J.; CASTRO, J. Tropos: A framework for requirements-driven software development. *Information Systems Engineering: State of the Art and Research Themes, Lecture Notes in Computer Science*, Springer, 2000.

[13] SILVEIRA, F. R. V. Uma abordagem fundamentada em agentes racionais para o teste de agentes racionais. Dissertação (Mestrado) — Ciência da Computação - Universidade Estadual do Ceará, 2013.

[14] WOOLDRIDGE, M. *An introduction to multiagent systems*. [S.l.]: John Wiley & Sons, 2009.