Towards a fault model for BDI agents: an initial study

Francisco J. P.Cunha, Elder Cirilo, Carlos Lucena
Laboratório de Engenharia de Software
Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Brasil
{fcunha,ecirilo,lucena}@inf.puc-rio.br

***Abstract*—Given the growing approaches based on the paradigm of multi-agent systems and hence the need to maintain the quality of the software produced, testing becomes an activity of a great importance. For the agent-oriented projects in the BDI architecture is not different. Thus, it is necessary for testing agents to have a suitable fault model that ensure the correctness of the proposed behavior. In this paper, we propose an initial analysis of a fault model. The structure of plans and goals of the agents is transformed into a sequence of actions in order to allow its verification and execution control.**

Keywords— fault model, tests in BDI agents, testability agents;

I. INTRODUÇÃO

Soluções de software baseadas em sistemas multi agentes são cada vez mais utilizadas. Agentes tem a capacidade de raciocinar usando a cooperação com outros agentes para alcançar metas estabelecidas [3]. Assim como nos paradigmas tradicionais de desenvolvimento, há a necessidade de garantir robustez e corretude, o que se dá através de testes [1]. Uma observação importante é que, testes em sistemas multi agentes não é uma tarefa trivial se comparada a sistemas tradicionais.

A proposta desse trabalho é realizar um estudo inicial sobre a testabilidade em sistemas de agentes BDI, identificando as características relevantes para elaboração de um modelo de falhas. Assim, analisamos quantitativamente a execução de um agente. No que segue, uma introdução da arquitetura BDI é apresentada na seção 2, na seção 3 é apresentado o mapeamento dos agentes para uma árvore de planos e metas e na seção 4 é analisada quantitativamente a execução. A conclusão é apresentada na seção 5.

II. BACKGROUND – ARQUITETURA BDI

Uma arquitetura de sucesso para desenvolver agentes deliberativos é o modelo BDI (Belief, Desire e Intention). [7] desenvolveu uma teoria de raciocínio prático humano que descreve o comportamento de maneira racional pelas noções de "Crenças", "Desejos" e "Intenções". A implementação do presente modelo substitui os dois últimos conceitos pelos conceitos de "metas" e "planos". Contudo, meta não é um conceito chave em sistemas BDI uma vez que são modeladas como eventos. Assim, estabelecer uma meta, corresponde a selecionar e executar um plano que possa manipular esse evento [5]. No restande deste trabalho consideramos os planos como manipulação de eventos [2]. Um plano é constituído por três partes: um evento padrão especificando o que é relevante, uma condição de contexto indicando em que situações o plano pode ser usado, e o corpo do plano [2].

Um ciclo de execução BDI é uma sequência como a descrita abaixo: (i) um evento ocorre; (ii) o agente determina um conjunto de instâncias de planos cujos eventos padrão correspondem ao evento gerador; (iii) o agente avalia as condições de contexto dos planos relevantes para gerar o conjunto de instâncias aplicáveis dos planos; (iv) uma das instâncias aplicáveis do plano é selecionada e executada; (v) se o corpo plano falhar, então um mecanismo de manipulação de falhas é acionado.

Este ciclo gera uma trilha de operações que compreende a sequência de ações com tentativas de execução e um indicador de sucesso ou fracasso. Existem algumas abordagens para lidar com o fracasso. Talvez a abordagem mais comum, e que é utilizada em diversas plataformas BDI existentes é a de selecionar um plano alternativo aplicável, e considerar apenas um evento como tendo falhado, quando não existem planos aplicáveis restantes. Ao determinar planos alternativos pode-se considerar o conjunto de planos aplicáveis existentes ou recalcular o conjunto de planos aplicáveis uma vez que a situação pode ter mudado desde que os planos foram determinados. Algumas plataformas BDI usam como tratamento de falhas, repetir os planos em caso de falha [2].

III. EXECUÇÃO BDI COMO ÁRVORE DE PLANOS E METAS

A execução de um agente BDI é um processo dinâmico que executa ações progressivamente. Na análise deste processo e, para torná-lo numa forma declarativa, aplicamos uma transformação de maneira que o comportamento do agente seja mapeado em uma árvore de planos e metas e que seja executado como uma sequência de ações. Os planos e metas podem ser visualizados como uma árvore onde cada meta tem como filhos os planos que lhes são aplicáveis e cada plano tem como filhos os seus subplanos [2]. Trata-se de um tipo de árvore "e/ou" onde cada meta é realizada por um dos seus planos ("ou") e cada plano precisa de todos os seus subplanos para ser alcançado ("e") [3]. A escolha do plano para cada meta na árvore não é um processo determinista. Além disso, ao considerar uma falha, precisa-se saber o que fazer para recuperação. Inicialmente, uma árvore de planos e metas é representada como um termo em Prolog seguindo uma gramática simples [3]. GPT é a abreviação de "Goal-Plan

Tree", AoGL é a abreviação de "Action or Goal List", e A é um símbolo.

```
⟨GPT⟩ ::= goal([]) | goal([⟨PlanList⟩])
⟨PlanList⟩ ::= ⟨Plan⟩ | ⟨Plan⟩, ⟨PlanList⟩
⟨Plan⟩ ::= plan([]) | plan([⟨AoGL⟩])
⟨AoGL⟩ ::= act(A) | ⟨GPT⟩ | act(A), ⟨AoGL⟩ | ⟨GPT⟩, ⟨AoGL⟩
```

Fig. 1 Uma gramática para transformação da árvore de planos e metas [2]

Como um exemplo, retirado do trabalho de Winikoff et al. [2], uma meta com dois planos aplicáveis, cada um contendo uma única ação, é representado pelo termo: goal ([plan ([act (a)]), plan ([act (b)])]). Um predicado não determinista exec é definido onde seu primeiro argumento (entrada) é a árvore de planos e metas e o segundo argumento (saída) é uma sequência de ações. Consideremos que a árvore de planos e metas contenha somente instâncias de planos aplicáveis. Assim, para transformar um "nó meta" em uma sequência de ações, selecionamos uma de suas instâncias aplicáveis do plano. O plano selecionado é transformado em uma sequência de ações (linha 2). Quaisquer uns dos planos aplicáveis podem ser escolhidos e não apenas o primeiro. Se o plano selecionado é executado com sucesso então o resultado do trace é uma sequência de ações para a execução de uma meta (linha 3). Caso contrário, é executada a recuperação de falhas (linha 10), que é feita tomando os planos restantes, ou seja, excluindo o plano que já foi tentado. A sequência de ações resultantes é anexada à sequência de ações do plano de falha para obter uma sequência completa de ação para o objetivo. Especificamente, um plano aplicável é selecionado e executado, e se for bem sucedido, então a execução para. Se não for bem sucedido, então um plano alternativo é selecionado e a execução continua, ou seja, as sequências de ações são anexadas.

```
1  exec(goal([]),[]).
2  exec(goal(Plans), Trace) :- remove(Plans,Plan,Rest), exec(Plan,Trace1),
3      (failed(Trace1) -> recover(Rest,Trace1,Trace) ; Trace=Trace1).
4  exec(plan([]), []).
5  exec(plan([Step|Steps]), Trace) :- exec(Step,Trace1),
6      (failed(Trace1) -> Trace=Trace1 ; continue(Steps,Trace1,Trace)).
7  exec(act(Action), [Action]).
8  exec(act(Action), [Action,fail]).
9  failed(Trace) :- append(X,[fail],Trace).
10 recover(Plans,Trace1,Traces) :-
11     exec(goal(Plans),Trace2), append(Trace1,Trace2,Traces).
12 continue(Steps,Trace1,Trace) :- exec(plan(Steps),Trace2),
13     append(Trace1,Trace2,Trace).
14 % remove(A,B,C) iff removing element B from list A leaves list C
15 remove([X|Xs],X,Xs).
16 remove([X|Xs],Y,[X|Z]) :- remove(Xs,Y,Z).
```

Fig. 2 Mapeamento da árvore de planos e metas em uma sequência de ações [2]

IV. Comportamento dos Agentes BDI

É importante considerar o número de possibilidades de comportamentos existentes para um agente BDI que está tentando atingir uma meta. Usando o mapeamento anterior, vemos a execução de um agente BDI como a transformação de uma árvore de planos e metas para uma sequência de ações. Assim, a respeito do número de possibilidades para o comportamento de agentes BDI, derivam fórmulas que permitem calcular o número de comportamentos, de sucesso e fracasso para uma determinada árvore de planos e metas [3]. Assumindo que, todas as subárvores de um nó plano ou meta têm a mesma estrutura podemos definir a profundidade de uma árvore de planos e metas como o número de níveis de "nós-meta" que ele contém. Uma árvore de profundidade igual a 0 é um plano sem submetas, enquanto que uma árvore com profundidade d > 0 ou é um nó plano com filhos que são nós-meta de profundidade d, ou um nó meta com filhos que são planos em profundidade d - 1. Assumimos que todos os planos de profundidade d > 0 têm k submetas e que todas as metas têm j instâncias aplicáveis do plano. Este pode ser o caso de cada meta ter j planos relevantes, cada qual resultando em exatamente um plano aplicável, mas também pode ser o caso de outras formas, por exemplo, uma meta pode ter 2 planos relevantes, metade dos quais são aplicáveis na situação atual.

A seguinte terminologia foi utilizada nas seções abaixo: (i) pressupomos que a estrutura da subárvore com raiz em um nó plano ou meta é determinada unicamente por sua profundidade e, portanto, podemos denotar uma meta ou plano em profundidade d como g_d ou p_d, respectivamente; (ii) usamos $n^{\checkmark}(x_d)$ para identificar o número de caminhos de execução bem sucedida de uma árvore de profundidade d e com raiz em x; (iii) analogamente, usamos $n^{\times}(x_d)$ para denotar o número de caminhos de execução mal sucedidas de uma árvore de profundidade d com raiz x; (iv) estendemos a notação para as sequências $x_1;...;x_n$ onde cada x_i é uma meta ou ação e ";" denota uma composição sequencial. Abreviamos a sequência de n ocorrências de x por x^n.

A. *Caso Base: Comportamento com execuções de sucesso*

Iniciamos o processo calculando o número de caminhos bem sucedidos na árvore de planos e metas na ausência de falhas [6]. O número de caminhos possíveis para a alcançar uma meta é a soma do número de caminhos em que seus filhos podem ser alcançados. Por outro lado, o número de caminhos para se alcançar um plano é o produto do número de maneiras em que os seus filhos podem ser alcançados [2]. Sendo uma árvore com raiz g, assumimos que cada um dos seus j filhos podem ser alcançados de n diferentes maneiras, então, ao selecionar um dos filhos o número de maneiras na qual g pode ser alcançado é jn. Da mesma forma, para uma árvore com raiz p ("plan"), assumimos que cada um dos seus filhos k filhos podem ser alcançados de n maneiras diferentes, então, ao executar todos os seus filhos, o número de maneiras na qual p pode ser executado é n^k. Um plano sem filhos pode ser executado exatamente de uma maneira. Representamos o cenário descrito acima respectivamente pelas equações: $n^{\checkmark}(g_d) = jn^{\checkmark}(p_{d-1})$; $n^{\checkmark}(p_0) = 1$; $n^{\checkmark}(p_d) = n^{\checkmark}(g_d)^k$.

B. *Adicionando Falha*

Estendendo a análise para incluir falhas, determinamos o número de execuções mal sucedidas. Nenhum mecanismo de tratamento de falhas será considerado. Para determinar o número de execuções mal sucedidas, precisamos saber onde cada falha pode ocorrer. Em sistemas BDI há dois lugares onde as falhas ocorrem: quando uma meta não possui instâncias de planos aplicáveis e, quando uma ação dentro do corpo de um plano falha [2]. Neste trabalho consideramos somente metas com instâncias aplicáveis. Logo, um plano pode falhar devido à falha de quaisquer umas das ações e submetas presentes no corpo do plano. Mais especificamente, um plano falha se a tentativa de executar sequencialmente seu corpo falha.

Generalizando, podemos assumir que há um número de ações antes, depois e entre as submetas de um plano. Um plano

sem submetas é considerado consistente e de uma única ação. Assim, o número de caminhos mal sucedidos de uma meta é definido por $n\times(gd) = j$, de um plano que é uma folha na árvore $(d=0)$ por $n\times(p0) = \varphi$ e um plano $(d > 0)$ por $\varphi + (n\times(gd) + \varphi n\checkmark(gd)) * (n\checkmark(gd) - 1/ n\checkmark(gd) – 1)$ [2].

C. *Adicionando tratamento de falha*

Uma forma comum de lidar com as falhas é responder a falha de um plano, tentando aplicar um plano alternativo ao evento gerador. Como resultado, é mais difícil ocorrer uma falha. A única maneira é se todos os planos falharem. O efeito da adição do tratamento de falha é converter possíveis falhas para sucessos, ou seja, uma execução que de outra forma seria mal sucedida, é estendida para uma longa execução que pode ser bem sucedida. O número de execuções de uma meta com j instâncias de planos aplicáveis e φ ações no corpo do plano, pode ser representado por: $n\times(g_d) = j!n\times(p_{d-1})^j$; $n\times(p_0) = \varphi$; $n\times(p_d) = \varphi + (n\times(g_d) + \varphi n\checkmark(g_d))(n\checkmark(g_d)^k -1/ n\checkmark(g_d) -1)$.

D. *A probabilidade da execução falhar*

A introdução do tratamento de falhas torna a possibilidade de falha na execução muito menor. Quando olhamos unicamente para o número de possibilidades de falhas, verificamos que o número de possibilidade de ocorrer uma falha é grande. Acontece que, o mecanismo de tratamento de falhas reduz drasticamente esse número devido a probabilidade de uma falha, de fato, ocorrer.

E. *Análise dos números e equações*

Analisando as equações apresentadas verificamos que percorrer exaustivamente cada caminho possível na árvore de planos e metas é, de fato, inviável. A medida que a profundidade e largura da árvore aumentam, o número de possibilidades de caminhos cresce exponencialmente. Para dimensões de uma árvore pequena, talvez seja possível fazer algum tipo de verificação, mas sem dúvida, trata-se de uma solução não-escalável.

V. CONCLUSÃO

Como contribuição, o presente trabalho revisitou a literatura relacionada a testabilidade em SMA de agentes BDI. Essa revisão inicial revelou que, o número de comportamentos possíveis na execução de agentes BDI de fato cresce à medida que a profundidade e a largura das árvores de planos e metas aumentam. Uma observação interessante é que, a introdução de tratamento de falha faz uma diferença significativa no número de comportamentos. Ao considerar o teste do sistema como um todo, concluímos sobre a testabilidade de sistemas de agentes BDI que, de acordo com as equações apresentadas, tentar obter a garantia de correção do sistema por meio de testes do sistema como um todo, não é viável. Na verdade, a situação é ainda pior quando consideramos não apenas o número de possíveis execuções, mas também a probabilidade de falha. Então, o teste do sistema de agentes BDI parece ser impraticável, nessas condições. Sobre testes unitários e testes de integração, apesar dos testes de unidade e integração serem relevantes, nem sempre é clara a forma de aplicá-los utilmente para sistemas de agentes. Para os agentes BDI, ao testar uma submeta, pode ser difícil de assegurar que o teste abrange todas as situações em que podem ser tentados. Igualmente, ao testar um agente sem o resto do sistema (incluindo outros agentes) pode ser difícil garantir a cobertura das diferentes possibilidades.

No entanto, a conclusão parece ser a de que SMA BDI são verificáveis através de testes quando restringimos certas caracteristica da representação do comportamento dos agentes. Neste sentido, constatamos que existem diveras lacunas em aberto e, assim, a existência de uma área promissora de pesquisa. Podemos considerar como tópicos de pesquisa: (i) a investigação de mecanismos para geração de planos de testes automatizados; (ii) rastreamento e mapeamento das condições de contexto e trocas de mensagens para a sequência de ações geradas; (iii) análise de um modelo de falhas; e (iv) estudo de viabilidade de extensão do framework JAT para utilização do modelo de falhas proposto.

REFERENCES

[1] Sommerville, I., Software Engineering (Sixth edition), Addison Wesley (2000).

[2] Winikoff, M., Cranefield, S., On the testability of BDI agent systems. Information Science Discussion Paper 2008/03, University of Otago, Dunedin, New Zealand, 2008.Disponível em http://www.business.otago.ac.nz/infosci/pubs/papers/dpsall.htm.

[3] Sudeikat, J., Validation of BDI Agents. In: Bordini, R.H., Dastani, M.M., Dix, J., El Fallah Seghrouchni, A. (eds.) PROMAS 2006. LNCS (LNAI), vol. 4411, pp. 185–200. Springer, Heidelberg (2007).

[4] Low, C.K., Chen, T.Y., Rönnquist, R.: Automated test case generation for BDI agents. In: Autonomous Agents and Multi-Agent Systems, vol. 2, pp. 311–332 (1999).

[5] Nunes, I., Lucena, C.J.P., Luck, M. (2011), BDI4JADE: a BDI layer on top of JADE, in Louise A. Dennis, Olivier Boissier and Rafael H. Bordini, ed., Ninth International Workshop on Programming Multi-Agent Systems (ProMAS 2011), Taipei, Taiwan, pp. 88-103.

[6] Padgham, L., Winikoff, M., Developing Intelligent Agent Systems: A Practical Guide. John, Wiley and Sons (2004).

[7] Bratman, M.: Intentions, Plans, and Practical Reason. Harvard Univ. Press (1987).