I workshop de Comunicação sem Fio
4/6/99

Especificação de Agentes Móveis Usando Máquinas de Estado Abstratas

Marco Túlio de Oliveira Valente[1] Roberto da Silva Bigonha[2]
Marcelo de Almeida Maia[3] Antônio Alfredo F. Loureiro[4]

[1]Pontifícia Universidade Católica de Minas Gerais, mtov@dcc.ufmg.br
[2]Universidade Federal de Minas Gerais, bigonha@dcc.ufmg.br
[3]Universidade Federal de Ouro Preto, marcmaia@dcc.ufmg.br
[4]Universidade Federal de Minas Gerais, loureiro@dcc.ufmg.br

Resumo

Neste artigo propõe-se um método para especificação formal de agentes móveis usando Máquinas de Estado Abstratas (ASM) [8], um formalismo que vêm sendo usado com sucesso para modelar diversos outros tipos de sistemas computacionais. O método proposto é baseado no Cálculo de Ambientes [6], um cálculo de processos projetado especificamente com o objetivo de expressar mobilidade. Mostra-se no trabalho que as abstrações fundamentais para computação móvel introduzidas pelo Cálculo de Ambientes podem ser mapeadas com facilidade em ASM. A fim de comprovar a viabilidade do método proposto, apresenta-se como estudo de caso a especificação em ASM de um agente móvel para comércio eletrônico.

Abstract

In this paper we present a formal specification method for mobile agents using Abstract State Machines (ASM) [8], a formalism which has already been successfully applied to model a variety of systems. The method is based on the Ambient Calculus [6], a process calculus developed to express mobility. In the work we show how the fundamental abstractions of the Ambient Calculus can be expressed in ASM without difficulty. In order to exhibit the feasibility of the proposed method, we also show as a case study an ASM specification of a mobile agent for electronic commerce.

1 Introdução

Na década de 90, assistiu-se a uma forte expansão da Internet e particularmente da *World Wide Web (WWW)*, a qual transformou-se em uma grande infra-estrutura computacional cobrindo todo o planeta. No entanto, atualmente a WWW é utilizada majoritariamente como um sistema cliente/servidor puro, onde os clientes, representados por *browsers Web*, solicitam informações (textos, imagens etc) localizadas em um servidor. A principal vantagem deste paradigma é a sua simplicidade, tanto para o usuário final, como para aqueles encarregados de disponibilizá-lo e de mantê-lo. Provavelmente esta simplicidade foi a grande responsável pela sua popularização. Por outro lado, ele tem como desvantagem o fato de não aproveitar adequadamente todo o potencial computacional da rede.

O desenvolvimento da linguagem Java foi a primeira iniciativa para reverter esta desvantagem. Com Java disponibilizou-se uma infra-estrutura comum, chamada de *Java Virtual Machine* (JVM), capaz de executar localmente programas (*applets*) embutidos em páginas *Web*. Em Java, mobilidade de código foi então utilizada como uma solução para permitir a realização de computações nos nós clientes da rede.

Mais recentemente, agentes móveis vêm sendo propostos como outra alternativa para ampliar a utilização computacional da *Web* [10, 11]. Agentes móveis caracterizam-se por permitir a mobilidade não apenas do código de uma aplicação, mas também do estado de sua execução. Assim, uma computação não mais se restringe a um único nó da rede, podendo migrar para outros nós, onde, por exemplo, encontram-se localmente disponíveis os recursos de que ela necessita para execução de sua tarefa.

À medida que esta forma de computação vem se tornando realidade, surge a necessidade de se desenvolver métodos formais para especificação dos sistemas desenvolvidos segundo este paradigma. Através destes formalismos, será possível entender corretamente o comportamento de agentes móveis, provar propriedades a respeito dos mesmos e também propor novas linguagens de programação e ambientes que apoiem o seu desenvolvimento. Recentemente, alguns métodos formais vêm sido propostos com este fim. Dentre estes, o cálculo de ambientes [5, 6] destaca-se por ter sido proposto originalmente com o objetivo de especificar computação móvel, ao passo de que os outros são extensões ou variações de modelos originalmente desenvolvidos tendo em vista a modelagem de sistemas distribuídos.

Neste trabalho, procura-se mostrar que as abstrações fundamentais para especificação de mobilidade propostas no cálculo de ambientes podem ser mapeadas sem grandes dificuldades para Máquinas de Estado Abstratas (ASM) [8, 9, 16]. ASM é um formalismo que vem sendo utilizado com sucesso para especificar diversos tipos de sistemas computacionais, incluindo linguagens de programação, arquiteturas, sistemas distribuídos e sistemas de tempo real. Além de já constituir um método formal estabelecido, especificações ASM possuem a vantagem de serem executáveis.

Na seção 2 deste trabalho, conceitua-se computação e agentes móveis e relacionam-se as vantagens propiciadas pela introdução deste paradigma de computação na *Web*. Na seção 3, descrevem-se os modelos formais que vêm sendo atualmente propostos para computação móvel. A seção 4 apresenta as Máquinas de Estado Abstratas e a seção 5 descreve como as abstrações do cálculo de ambientes podem ser mapeadas em ASM. Na seção 6 do trabalho, mostra-se um estudo de caso consistindo na especificação de um agente móvel para comércio eletrônico em ASM. Finalmente, a seção 7 apresenta as conclusões do artigo e sugestões para trabalhos futuros.

2 Agentes Móveis

O termo agente é usado em diferentes áreas da ciência da computação, incluindo Inteligência Artifical, Bancos de Dados e Recuperação de Informação. Particularmente, neste trabalho trataremos de agentes móveis, isto é, de processos que migram de um computador para outro da rede com o objetivo de executar uma tarefa específica [12]. Um agente móvel possui instruções, dados e um estado de execução. A execução de um agente móvel se dá de forma independente

da aplicação que o invocou. Algumas das características principais de agentes móveis são: habilidade de um agente interagir e cooperar com outros agentes, autonomia no sentido de que sua execução procede com nenhuma ou pouca intervenção da entidade que o disparou, execução em diferentes plataformas de *hardware* e *software* (interoperabiliade), capacidade de responder a eventos externos (reatividade) e capacidade de mover de uma estação para outra (mobilidade).

Embora não devam ser considerados como uma bala de prata capaz de resolver todos os tipos de problemas, agentes móveis vêm sendo avaliados como uma tecnologia capaz de tornar mais fácil o projeto, implementação e manutenção de sistemas distribuídos [10]. Uma principal vantagem do seu uso é a redução da carga e latência de rede. Normalmente, sistemas distribuídos requerem um grande volume de comunicação para realização de sua tarefa, o que gera um alto tráfego na rede. Agentes móveis permitem que esta interação seja "empacotada" e enviada ao nó destino, onde então ela se faz localmente. São também úteis para reduzir o volume de dados transportados pela rede. Por exemplo, quando grandes volumes de dados são armazenados em nós remotos, pode-se enviar um agente até o mesmo em vez de se transferir todos os dados até o nó origem.

No caso de dispositivos computacionais móveis (PDAs, *notebooks* etc), agentes móveis são particularmente úteis devido ao fato de executarem de forma assíncrona e autônoma do nó de onde foram disparados [12]. Por exemplo, durante uma rápida conexão, um cliente móvel pode invocar um agente e então desconectar-se. O agente prossegue então independentemente com o objetivo de executar sua tarefa. Quando a tarefa encontrar-se terminada, o agente espera por uma conexão com o cliente móvel para enviar o resultado de sua computação.

Atualmente, diversas aplicações vêm sendo relacionadas como beneficiárias diretas deste novo modelo de computação. Dentre elas, podemos citar comércio eletrônico, recuperação de informação, gerência de redes de telecomunicações, aplicações de *workflow* e *groupware* e processamento paralelo [11].

Vários ambientes vêm sendo propostos para desenvolvimento de agentes móveis, sendo que grande parte deles utiliza a linguagem Java [1]. Tal escolha parece ser natural, pois Java já incorpora alguns requisitos indispensáveis a este tipo de sistema, como, por exemplo, mobilidade de código, um ambiente seguro para execução e recursos para serialização de objetos e para carregamento dinâmico de classes. Dentre os ambientes já disponíveis, destacam-se os seguintes: Obliq [3, 4], Aglets [2], Odyssey (anteriormente chamado de Telescript) [14], Voyager [17] e D´Agent (anteriormente conhecido por Agent Tcl) [7].

3 Cálculo de Ambientes

Sendo a *Web* em grande medida um sistema distribuído global, existem propostas para se expressar mobilidade na *Web* utilizando os formalismos já existentes para especificação de sistemas distribuídos e concorrentes. Dentre estes, um dos mais utilizados é o π-cálculo [13, 15], o qual se baseia na noção de processos comunicando-se através de canais, com a possibilidade de se criar novos canais e de se enviar canais através de outros canais.

As propostas que defendem o uso de π-cálculo para especificação de sistemas móveis partem do pressuposto de que o conjunto de canais de um processo é um indicativo de sua localização e que, portanto, mobilidade pode ser descrita como uma mudança no número de canais do

processo. No entanto, essa definição não reflete o conceito de mobilidade de agentes tal como descrito na seção 2, o qual está relacionado com uma mudança no ambiente de execução do processo e não no seu número de canais. Nesse sentido, parece mais adequado afirmar que no π-cálculo se têm mobilidade de canais e não de processos.

Baseado exatamente nesta constatação é que foi proposto o cálculo de ambientes [6, 5], um cálculo de processos cujo foco principal é a mobilidade dos mesmos. Um ambiente é um local delimitado que possui um nome e que pode conter processos e subambientes. Um ambiente pode ainda se mover para dentro ou para fora de outro ambiente. Sendo um local com fronteiras definidas, fica fácil determinar o que move junto com um ambiente. No cálculo de ambientes, portanto, mobilidade está relacionada com a noção de cruzar fronteiras, as quais delimitam ambientes, sendo que estes por sua vez estão organizados de forma hierárquica, dando origem a uma estrutura de árvore.

Um ambiente é denotado por $n[P]$, onde n é o nome do ambiente e P é o processo em execução no seu interior. O processo P pode ser a composição em paralelo de diversos processos. Existem operações para alterar a estrutura hierárquica dos ambientes, as quais têm sua aplicação restringida por capacidades. A notação $M.P$ denota um processo que executa uma ação regulada pela capacidade M e então prossegue sua execução como se fosse o processo P. Existem três tipos de capacidade: para entrar, sair e abrir um ambiente.

A capacidade *in* m, usada em uma ação da forma *in* $m.P$, instrui o ambiente que a cerca a entrar em um ambiente vizinho de nome m. Se não existir tal ambiente, a operação fica bloqueada até que o mesmo passe a existir. Se existir mais de um ambiente com este nome, um deles é escolhindo aleatoriamente. Já a capacidade *out* m, usada em ações da forma *out* $m.P$, instrui o ambiente que a contém a sair de seu ambiente pai, cujo nome deve ser m. Se tal ambiente não se chama m, a operação fica bloqueada até que o nome do mesmo passe a ser este. Por fim, a capacidade *open* m, usada em uma ação da forma *open* $m.P$, abre as fronteiras do ambiente vizinho de nome m. Como nas ações anteriores, a operação fica bloqueada até que exista um ambiente vizinho com este nome.

Existem ainda ações para replicação de um processo P, denotada por $!P$ e para criação de nomes em um escopo P, denotada por $(\upsilon\ n)P$.

No cálculo de ambientes define-se então precisamente o conceito de localização de um processo e sua capacidade de mover de um local para outro, abstração esta que reflete com fidelidade o comportamento dos sistemas móveis descritos na seção 2.

4 Máquinas de Estado Abstratas (ASM)

Máquinas de Estado Abstratas (em inglês *Abstract State Machines* ou ASM) constituem um modelo computacional capaz de especificar qualquer algoritmo seqüencial em seu nível natural de abstração [8, 9, 16]. Numa ASM, um estado S é uma álgebra definida por um vocabulário Υ de nomes de funções e relações, um conjunto X chamado de superuniverso e uma função de interpretação dos nomes do vocabulário em funções de $X^* \rightarrow X$. Regras de transição são usadas para modificar a interpretação dos nomes de função de um estado para outro. Assim, a execução de um programa ASM consiste no disparo de regras de transição que fazem com que a máquina mude sucessivamente de estado. Por esta razão, ASM já foram chamadas de Álgebras

Evolutivas.

Em ASM existem três regras básicas de transição: regra de atualização, regra condicional e regra de construção de blocos.

Uma regra de atualização é da forma $f(t_1, \ldots, t_j) := t_0$, onde j é a aridade de f e $t_0, \ldots, t_j$ são termos. O disparo desta regra em um estado S, onde os termos $t_0, \ldots, t_j$ são avaliados respectivamente como $a_0, \ldots, a_j$, dá origem a um novo estado S' onde o conteúdo da função interpretada pelo nome f no ponto $a_1, \ldots, a_j$ é igual a a_0.

Uma regra condicional possui a seguinte forma: **if** φ **then** R_1 **else** R_2 **endif**, onde φ é um termo booleano e R_1 e R_2 são regras. A semântica desta regra é óbvia: se o termo φ avaliar em *true*, então o próximo estado é o resultante do disparo da regra R_1; caso contrário, é o resultante do disparo da regra R_2. Existe também o comando **elsif**, usado para aninhar expressões condicionais, eliminando assim a repetição de comandos **endif**.

Por último, uma regra de construção de bloco é da forma $R_1, \ldots, R_n$ e possui a seguinte semântica: o próximo estado é o resultante do disparo de todas as regras R_i *em paralelo*, isto é, a ordem de $R_1, \ldots, R_n$ não é relevante na execução do bloco. Não existe em ASM composição seqüêncial de regras, ou seja, uma regra da forma $R_1; R_2$, onde primeiro executa-se R_1 e então R_2. O motivo é fazer com que programas ASM descrevam um algoritmo passo a passo.

Uma especificação ASM contém a definição de um estado inicial S_0 e uma regra de transição R que define as mudanças de estado. A execução de uma especificação é uma seqüência de estados $\langle S_n : n \geq 0 \rangle$, onde um estado S_i é obtido executando a regra R em S_{i-1}.

5 Abstrações para Computação Móvel em ASM

A abstração principal para computação móvel proposta no trabalho de Cardelli foi a noção de ambiente, definido como um "local delimitado onde acontecem computações"[5]. Em ASM, também existe esta noção de ambiente, só que com a finalidade de prover funções externas utilizadas pela máquina em sua computação [8]. Funções externas são usadas como um oráculo: fornecidos os seus argumentos, o oráculo devolve o valor, o qual não necessariamente precisa ser o mesmo de uma chamada contida em um passo anterior do algoritmo, mesmo que os argumentos sejam idênticos. No entanto, em um mesmo passo, os resultados forncecidos pelo oráculo devem ser consistentes. Dentre outras aplicações, funções externas são utilizadas para modelar interação com o ambiente (entrada/saída), para expressar não determinismo e para prover ocultamento de informação.

Neste trabalho, propõe-se que ambientes sejam usados não apenas como repositório de funções externas, mas também como ponto de referência para localização da computação realizada por uma ASM. Com isso, a exemplo do cálculo de Cardelli, propõe-se que mobilidade em ASM seja caracterizada como a capacidade de deslocar uma computação, isto é, o estado de uma ASM, de um ambiente para outro.

Propõe-se que, além de funções externas, um ambiente tenha também as seguintes características:

- Um ambiente possui um nome, usado para controlar o acesso ao mesmo.
- Um ambiente possui um conjunto de ASMs, todas elas em execução.

- Um ambiente possui um conjunto de subambientes, isto é, ambientes são organizados hierarquicamente.
- Um ambiente provê uma operação atômica chamada de *move* para todas as ASMs em execução em seu interior.

Suponha que a máquina M esteja em execução no ambiente m e que seu estado atual seja S_i. Então a execução de uma regra de transição contendo a operação *move* n faz com que o próximo estado da máquina, isto é, S_{i+1}, esteja localizado no ambiente n. Caso este ambiente não esteja disponível, a execução de M fica bloqueada em S_i até que a transição para S_{i+1} possa ser completada. Esta operação equivale à capacidade *in* n do cálculo de ambientes, descrito na seção 3. Ela é também equivalente a instruções disponíveis nos sistemas mencionados na seção 2, como, por exemplo, *hop* (de Obliq), *go* (de Telescript) e *dispatch* (de Aglets).

A operação *move* adiciona então a noção de mobilidade de estado a uma especificação ASM, onde o conceito de estado é o mesmo da definição original do formalismo, isto é, um vocabulário Υ, um superuniverso X e uma função de interpretação $Val_S : \Upsilon \rightarrow X^* \rightarrow X$. Um *move* apenas altera a localização de um estado, isto é, o ambiente a que o estado pertence. Com isso, em ASM o *binding* entre a chamada de uma função externa e a sua definição em algum ambiente passa a ser dinâmico. Veja ainda que não se exige que funções externas de mesmo nome, mas pertencentes a ambientes distintos, possuam a mesma semântica. Apenas exige-se que elas tenham a mesma assinatura.

A computação de uma ASM sempre acessa as funções externas do ambiente corrente de execução. Caso se migre uma ASM para um ambiente que não ofereça uma função externa utilizada pelo programa, a execução da máquina será abortada quando esta função for referenciada em alguma transição.

A fim de que a execução de uma ASM possa consultar sua localização corrente, propõe-se ainda que esteja disponível nas especificações a função externa de zero argumentos *here*, a qual retorna o ambiente corrente de execução da máquina.

6 Estudo de Caso: Agente Móvel para Comércio Eletrônico

Mostra-se abaixo a especificação de um agente móvel simples para comércio eletrônico, o qual pesquisa o preço de um livro em um conjunto de livrarias da Internet. Supõe-se que o agente inicia sua execução em seu ambiente de origem, que chamaremos de *Origem*, e então visita uma série de livrarias eletrônicas, cada uma delas representada por um ambiente. Em cada livraria (ambiente), o agente (computação ASM) consulta e armazena em seu estado o preço do livro. Após visitar a última livraria, o agente retorna ao ambiente de origem, onde verifica em qual livraria o livro possui o menor preço.

Ambient Origem;

ASM PesquisaPreçoLivros;

External

```
total_livros: Integer;          // Total de livros à venda na livraria
lista_livros (Integer): String; // Lista com nome dos livros à venda
lista_preços (Integer): String; // Lista com preço dos livros à venda
```

Vocabulary

```
Ambient;                        // Universo de nomes de ambientes
nome_livro:  String;            // Livro a ser pesquisado
livrarias:   List of Ambient;   // Livrarias a serem visitadas
preço:       Ambient → Real;    // Preço do livro em cada livraria
situação:    enum { inicial, viajando, pesquisando, achou, não_achou, final };
índice:      Integer;
```

Init

```
livrarias:= [Amazon, Bookpool, BarnesAndNobles ];
situação:= inicial ;
```

Rule

```
if (situação = inicial) then
  situação:= viajando,
  livrarias:= tail (Livrarias),
  move head (livrarias)
elsif (situação = viajando) then
  situação:= pesquisando,
  índice:= 1
elsif (situação = pesquisando) then
  if índice > total_livros then
    situação:= não_achou
  elsif lista_livros (índice) = nome_livro then
    preço (here):= lista_preços (indice),
    situação:= achou
  endif,
  índice:= índice + 1
elsif (situação = achou) or (situação = não_achou) then
  livrarias:= tail (livrarias),
  if head (livrarias) = [] then
    situação:= final,
    move Origem
  else
    situação:= viajando,
    move head (livrarias)
  endif
```

elsif (situação = final)
// Transições para pesquisar menor preço encontrado na "viagem"

End;

Analisando esta especificação, poderia ser argumentado que a operação *move* não é necessária, podendo ser eliminada de uma especificação onde um parâmetro extra seria acrescentado a toda função externa com o objetivo de identificar o seu ambiente. No entanto, esta estratégia assumiria como válida a existência de um estado global ao sistema, o que na Internet ou na *Web* é equivalente a assumir a existência de um "Grande Irmão" gerenciando todo o sistema. Como sabemos que tal suposição não é válida, para se acessar os serviços de um ambiente deve-se explicitamente tentar cruzar suas fronteiras, assumindo todos os riscos inerentes a uma ação como essa (por exemplo, o ambiente não estar disponível ou ter permissão negada para atravessar sua fronteira) [5]. Daí a necessidade da função *move*.

7 Conclusões

Com o surgimento de agentes móveis na Internet, torna-se importante o desenvolvimento de métodos formais que capturem as abstrações principais utilizadas neste paradigma de computação. Tais métodos permitirão um entendimento rigoroso do comportamento de sistemas móveis, a prova de propriedades a respeito dos mesmos e a proposta de novas linguagens e ambientes que facilitem o seu desenvolvimento. Neste trabalho, apresentou-se um método para especificação formal de agentes móveis utilizando Máquinas de Estado Abstratas (ASM) e inspirado nas abstrações principais para computação móvel propostas pelo Cálculo de Ambientes.

Em relação ao Cálculo de Ambientes, a especificação de agentes móveis em ASM, de acordo com o método proposto neste trabalho, apresenta as seguintes vantagens:

- ASM é um método formal de fácil entendimento, exigindo pouca bagagem matemática de seus usuários. Apenas conhecimentos básicos de algoritmos e teoria de conjuntos são suficientes para se produzir uma especificação ASM.
- ASM já foi usada com sucesso para especificar uma variedade de sistemas computacionais, incluindo linguagens de programação, sistemas distribuídos, sistemas de tempo real e arquiteturas. O objetivo principal deste trabalho é fazer com que ASM passe também a ser utilizada na especificação de sistemas móveis.
- Especificações ASM são diretamente executáveis, permitindo que o usuário não apenas especifique, mas também simule o funcionamento do sistema real. Já existem alguns interpretadores para ASM e outros encontram-se em desenvolvimento [16].

Especificação formal de sistemas móveis, sendo uma área de pesquisa ainda recente, oferece diversas possibilidades de trabalhos futuros, abordando temas que não foram tratados neste trabalho. Dentre eles, inclui-se segurança, comunicação entre agentes móveis, tratamento de exceções e sistemas de tipos para controlar mobilidade.

Referências

[1] Agent Society. *Agent Society Home Page.* http://www.agent.org.

[2] IBM Corporation. *Aglets Home Page.* http://www.trl.ibm.co.jp/aglets.

[3] Bharat, K. and Cardelli, L. *Migratory Applications.* Proceedings of the ASM Symposium on User Interface Software and Technology, 1995.

[4] Cardelli, L. *A Language with Distributed Scope.* Computing Systems, 8(1):26-59, MIT Press, 1995.

[5] Cardelli, L. *Abstractions for Mobile Computations.* Technical Report, Microsoft Research, 1999.

[6] Cardelli, L. and Gordon, A.D. *Mobile Ambients.* In *Foundation of Software Science and Computational Structures*, Maurice Nival (ed.), Lecture Notes in Computer Science 1378, Springer, 1998.

[7] Dartmouth College. *D'Agent Home Page.* http://agent. cs.dartmouth.edu/

[8] Gurevich, Y. *Evolving Algebras 1993: Lipari Guide.* In *Specification and Validation Methods.* E. Börger (ed.), Oxford University Press, 1995.

[9] Gurevich, Y. *Sequential ASM Thesis.* Technical Report MSR-TR-99-09, Microsoft Research, 1999.

[10] Lange, D.B. *Mobile Objects and Mobile Agents: The Future of Distributed Computing ?* Proceedings of The European Conference on Object-Oriented Programming, 1998.

[11] Lange, D.B. and Oshima, M. *Seven Good Reasons for Mobile Agents.* Communications of the ACM, 42(3), March 1999.

[12] Mateus, G.R. e Loureiro, A.A.F. *Introdução à Computação Móvel.* Décima Primeira Escola de Computação, Julho 1998.

[13] Milner, R.J., Parrow, J. and Walker, D. *A Calculus of Mobile Process.* Parts 1-2, Information and Computation, 100(1), 1992.

[14] GenMagic Inc. *Odissey Home Page.* http://www.genmagic.com/technology/odyssey.html.

[15] Pierce, B.C. *Foundational Calculi for Programming Languages.* CRC Handbook of Computer Science and Enginnering, 1996.

[16] Tirelo, F., Maia, M.A., Di Iorio, V. e Bigonha, R.S. *Máquinas de Estado Abstratas.* III Simpósio Brasileiro de Linguagens de Programação, Maio 1999.

[17] ObjectSpace Inc. *Voyager Home Page.* http://www.objectspace.com/voyager.htm.