Aplicação de ASM na Especificação de Sistemas Móveis

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

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

Resumo

Neste artigo apresenta-se um método para especificação formal de sistemas móveis usando Máquinas de Estado Abstratas (ASM). O método proposto é baseado no Cálculo de Ambientes, 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 sistema móvel para comércio eletrônico.

Palavras chave: sistemas móveis, métodos formais, ASM

Abstract

In this paper we present a formal specification method for mobile systems using Abstract State Machines (ASM). The method is based on the Ambient Calculus, 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 system for electronic commerce.

Key words: mobile systems, formal methods, ASM

1 Introdução

Recentemente, sistemas móveis vêm sendo propostos como uma alternativa para ampliar a utilização computacional da *Web*. Nos sistemas desenvolvidos segundo este modelo, uma computação, incluido o seu estado e o seu código, pode se mover de um nodo para outro da rede, onde, por exemplo, estão localmente disponíveis os recursos de que ela necessita para realizar a sua tarefa. Argumenta-se que este modelo computacional pode contribuir para reduzir a carga e a latência da Internet e para tornar mais simples o desenvolvimento

de aplicações que operem desconectadas da rede [10]. Este último requisito é essencial em qualquer aplicação baseada na *Web* e também em aplicações desenvolvidos para computadores móveis [11]. Atualmente, já existem diversos pacotes para desenvolvimento de sistemas móveis [1], sendo a maioria deles baseada em Java. Dentre estes pacotes, podemos citar os seguintes: Obliq [2, 3], Aglets [9], Odyssey (anteriormente chamado de Telescript) [13], Voyager [15] e D´Agent (anteriormente conhecido por Agent Tcl) [6].

À medida que este modelo de computação vem se tornando realidade, surge a necessidade de se desenvolver métodos formais para especificação dos sistemas desenvolvidos segundo seus princípios. Através destes formalismos, será possível entender corretamente o comportamento de sistemas 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 [4, 5] destaca-se por ter sido proposto originalmente com o objetivo de especificar computação móvel[1].

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) [7, 14]. ASM é um formalismo que vem sendo recentemente utilizado com sucesso para especificar diversos tipos de sistemas computacionais, incluindo linguagens de programação, arquiteturas, sistemas distribuídos e sistemas de tempo real. Espera-se que o método descrito neste artigo viabilize a utilização de ASM também na modelagem de sistemas móveis.

Na seção 2 deste trabalho, descrevem-se as idéias básicas do Cálculo de Ambientes. A seção 3 apresenta as Máquinas de Estado Abstratas. A seção 4 mostra como as abstrações para computação móvel propostas pelo Cálculo de Ambientes podem ser utilizadas em ASM. Na seção 5, apresenta-se um esquema de mapeamento destas abstrações para a própria linguagem das ASM. Na seção 6 do trabalho, mostra-se um estudo de caso consistindo na especificação de um sistema móvel para comércio eletrônico em ASM e provam-se duas propriedades deste sistema. Finalmente, a seção 7 apresenta as conclusões do artigo e sugestões para trabalhos futuros.

2 Cálculo de Ambientes

O Cálculo de Ambientes é um cálculo de processos, inspirado no π-Cálculo [12], e que tem como objetivo principal a modelagem de computação móvel. Segundo Cardelli, 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

[1] No contexto deste trabalho, o termo computação móvel é usado como tradução da expressão inglesa *mobile computation*. Ressalte-se que existe ainda a expressão *mobile computing*, usada para designar a situação em que os elementos da rede é que são móveis, como no caso das redes sem fio.

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 escolhido 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.

No Cálculo de Ambientes de Cardelli, existem ainda ações para replicação de um processo P, denotada por $!P$ e para criação de nomes em um escopo P, denotada por $(\nu\ n)P$.

3 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]. Numa ASM, um estado S é uma álgebra definida por um vocabulário Υ de nomes de funções, um conjunto X chamado de superuniverso e uma função de interpretação Val_S 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, isto é, a função Val_S. Assim, a execução de um programa ASM consiste no disparo de regras de transição que fazem com que a máquina mude sucessivamente de estado.

Antes de introduzir as regras de transição, seja a definição das seguintes noções auxiliares: endereços, atualizações, e conjunto de atualizações. Um endereço l de um estado S é um par $(f, \bar{x})$, onde f é um símbolo de função, $\bar{x} \in X^n$ e n é a aridade de f. As funções podem ser: 1) dinâmicas: aquelas cuja interpretação pode ser modificada, 2) estáticas: aquelas cuja interpretação não pode ser modificada, 3) externas: aquelas cuja interpretação não é controlada pela máquina, mas sim pelo ambiente externo. Chamadas a funções externas com os mesmos argumentos podem retornar valores diferentes em transições distintas.

Uma atualização α sobre um estado S é um par (l, x), onde l é um endereço e $x \in X$. Um termo pode ser definido da seguinte forma: se f é uma função n-ária e $t_1, \ldots, t_n$ são termos, então $f(t_1, \ldots, t_n)$ é um termo. Se $Val_S(t) = u$, então disparar $\alpha = ((f, \bar{x}), t)$ no estado S transforma S em S' tal que $Val_{S'}(f(\bar{x})) = u$ e todos os outros endereços não são afetados. Um conjunto de atualizações $Updates(R, S)$ é uma coleção de atualizações sobre o estado S, coletadas da regra de transição R. O conjunto de atualizações é consistente se ele não contém duas atualizações α, α' tal que $\alpha = (l, x)$ e $\alpha' = (l, y)$ e $x \neq y$. Caso con-

trário, o conjunto de atualizações é inconsistente. Disparar um conjunto de atualizações sobre um estado S significa disparar *simultaneamente* todas suas atualizações e produzir um estado correspondente S'. Disparar um conjunto de atualizações inconsistente significa nada atualizar, ou seja, produzir um estado $S' = S$.

As transições são definidas pelas seguintes regras:

$R \quad ::= \quad f(t_1, ..., t_n)$ `:=` $t \quad | \quad R_1 ... R_k \quad | \quad$ `if` e `then` R_1 `else` R_2

onde e é um termo booleano.

As regras acima são, respectivamente, a regra de *atualização*, o *bloco* de regras e a regra *condicional*. A semântica de uma regra R é dada pelo conjunto de atualizações $Updates(R, S)$, ou seja, para disparar R sobre um estado S, dispare $Updates(R, S)$. Este conjunto de atualizações é definido indutivamente na estrutura de R:

1. se $R \equiv f(t_1, ..., t_n)$ `:=` t então $Updates(R, S) = \{(l, Val_S(t))\}$,
   onde $l = (f, (Val_S(t_1), ..., Val_S(t_n)))$, e $Val_S(t)$ é o resultado de interpretar t em S;
2. se $R \equiv R_1 ... R_k$ então $Updates(R, S) = \cup_{i=1}^{k} Updates(R_i, S)$;
3. se $R \equiv$ `if` e `then` R_1 `else` R_2, então $Updates(R, S)$ é definido como:

$$\begin{cases} Updates(R_1, S), & \text{se } Val_S(e) \text{ vale} \\ Updates(R_2, S), & \text{caso contrário.} \end{cases}$$

Dado um estado inicial S_0, uma *execução* é uma seqüência de estados S_0, S_1,... tal que o estado S_{i+1} é obtido como resultado de disparar a relação de transição (representada pelo conjunto de atualização da regra da especificação) em S_i.

Finalmente, seja a definição de uma *ASM multi-agente,* a qual contém vários *agentes* computacionais, os quais executam concorrentemente certos programas (chamados módulos). Uma ASM multi-agente consiste em conjunto finito de programas π_v (módulos) e um número finito de agentes a, tal que $Val(a) \in X$ e para algum nome de módulo v, $Mod(a) = v$, onde a função *Mod* representa a relação entre nomes de módulos e agentes. Existe uma função de zero argumentos *self* a qual permite a auto-identificação de agentes: *self* é interpretada como a por cada agente a.

Um agente a efetua uma transição de um dado estado S se o conjunto de atualizações correspondente a a é disparado em S resultando um novo estado S'. Assim, uma transição pode ser representada por um par (S, S'). Com este conceito de transição podemos definir a noção de uma *execução parcialmente ordenada* para as ASM multi-agentes como uma tripla (M, A, σ), onde:

1. $M = (T, <)$ é um conjunto parcialmente ordenado (*poset*) de transições em T de agentes, tal que se $a, b \in T$, $a < b$ significa que a ocorre antes de b.
2. A é uma função que impõe que o conjunto de transições $\{m : A(m) = a\}$ obedeça a relação de ordem total.
3. σ é uma função que dado um segmento inicial de M (possivelmente vazio) atribui a ele o estado S correspondente. Um segmento inicial de um M é uma subestrutura X de M tal que se $x \in X$ e $y < x$ em P então $y \in X$. Dado que X é uma subestrutura, $y < x$ em X se e somente se $y < x$ em P sempre que $x, y \in X$.
4. A condição de coerência: Se x é *um* elemento maximal em um segmento inicial finito X de M e $Y = X - \{x\}$, então $A(x)$ é um agente em $\sigma(Y)$, e $\sigma(X)$ é obtido de $\sigma(Y)$ disparando $A(x)$ em $\sigma(Y)$.

4 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"[4]. Em ASM, também existe esta noção de ambiente, só que com a finalidade de prover funções externas utilizadas pela máquina em sua computação [7]. 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 o estado de uma ASM (isto é, um vocabulário Υ, um superuniverso X e uma função de interpretação $Val_S : \Upsilon \to X^* \to X$) 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. Portanto, esta operação externa adiciona a noção de mobilidade de estado a uma especificação ASM, onde o conceito de estado é o mesmo da definição original do formalismo.

Uma vez que com mobilidade uma mesma computação ASM pode agora migrar de ambiente para outro, define-se que a associação entre a chamada de uma função externa e a sua definição em algum ambiente é dinâmica.

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

5 Formalização da Noção de Mobilidade em ASM

Nesta seção, mostra-se como as abstrações para computação móvel propostas na seção anterior podem ser especificadas na própria linguagem das ASM.

Intuitivamente, temos que um sistema móvel é definido por um conjunto de agentes derivados de seus respectivos módulos, os quais devem obedecer a certas restrições. Um ambiente é um elemento do superuniverso, o qual norteia um determinado agente sobre

sua localização dentro do sistema. As restrições que os módulos devem obedecer são as seguintes:

- Os agentes derivados de um módulo devem acessar somente o seu estado local. Note que esta restrição inexiste em ASM multi-agentes uma vez que estas consideram o estado como uma entidade global.
- Todas as funções externas dependem do ambiente no qual elas são executadas.
- Um ambiente pode estar inacessível e, neste caso, toda operação de movimentação para este ambiente inacessível deve ser bloqueada.

Na Figura 1 mostramos o mapeamento entre ASM móveis e ASM multi- agentes.

	ASM Móveis	ASM Multi-agentes
Vocabulário		
Funções externas	$f_{ext} : X^r \rightarrow X$	$f_{ext} : X^{r+1} \rightarrow X$
Funções especiais	`here` *nome_ambiente*	`here(self)` *nome_ambiente*
Termos e endereços		
Funções externas	$f_{ext}(t_1, \ldots, t_r)$	$f_{ext}(here, t_1, \ldots, t_r)$

Regras	`move(`*ambient*`)`	`if _up(ambient) then`
		`  here(self) := ambient`
		`else`
		`  _blocked(self) := true`
		`  _down(self) := ambient;`

Módulos	`P`	`if not _blocked(self)then`
		`  P`
		`else if _up(_down(self))`
		`  _blocked(self) := false`
		`  here(self):=_down(self);`

Figura 1: Mapeamento entre ASM Móveis e ASM Multi-agentes

Um nome de ambiente `a` é um nome de função de zero argumentos que pertence ao vocabulário de uma ASM móvel. Nomes de ambientes diferentes são interpretados como valores diferentes em um conjunto de ambientes $A \subset X$. Existe um nome de função `here`$:A$, que é interpretada como o ambiente corrente. Somente a regra `move` pode mencionar nomes de ambientes diferentes de `here`. Na tradução da regra `move` e de um módulo da ASM Móvel são utilizadas as seguintes funções auxiliares: `_up`: $A \rightarrow \{true, false\}$ que é uma função externa que informa se um ambiente está acessível ou não; `_blocked`: $X \rightarrow \{true, false\}$ que é uma função usada para indicar o bloqueio da execução de um agente; `_down`: $X \rightarrow A$ usada para armazenar o ambiente inacessível para o qual um agente tentou mover-se, mas não conseguiu.

Proposição 1 *Se um agente a executar uma regra* `move(amb)` *para um ambiente* `amb` *inacessível (*`not _up(amb)`*) o agente permanece bloqueado (*`_blocked(self)`*) até o ambiente se tornar disponível.*

Prova: Através da guarda introduzida no programa de um agente a, temos que o agente executa o programa `P` original em um estado S se e somente se $Val_S(\_blocked\ (a)) = false$. Assim que o agente a execute a regra `move`, caso o ambiente esteja inacessível, temos que $Val_S(\_blocked(a))$ é modificado para *true*. O valor $Val_S(\_blocked\ (a))$ volta a ser *false* se e somente se o ambiente desejado estiver acessível, implicando no desbloqueamento do programa de a. □

6 Estudo de Caso

Mostra-se abaixo a especificação de um sistema móvel para comércio eletrônico, o qual pesquisa o preço de um livro em um conjunto de livrarias da Internet. Supõe-se que o sistema 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 sistema (computação ASM) consulta e armazena em seu estado o preço do livro. Após visitar a última livraria, o sistema 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** *// regra 1*
    situação:= viajando, livrarias:= tail (livrarias), **move** (head livrarias)
elsif (situação = viajando) **then** *// regra 2*
    situação:= pesquisando, índice:= 1

```
elsif (situação = pesquisando) then                                   // regra 3
    if índice > total_livros then
        situação:= não_achou
    elsif lista_livros (índice) = nome_livro then
        preço (here):= lista_preços (índice), situação:= achou
    endif,
    índice:= índice + 1
elsif (situação = achou) or (situação = não_achou) then               // regra 4
    livrarias:= tail (livrarias),
    if head (livrarias) = [ ] then
        situação:= final, move Origem
    else
        situação:= viajando, move head (livrarias)
    endif
elsif (situação = final)                                              // regra 5
    // Transições para pesquisar menor preço encontrado na "viagem"
end;
```

A seguir provam-se duas proposições a respeito deste sistema.

Proposição 2 *A pesquisa realizada pelo sistema em uma livraria sempre termina.*

Prova: Deve-se provar que se $Val_{S_i}(situação) =$ `pesquisando`, então existe um $j > i$, tal que $Val_{S_j}(situação) =$ `final` ou $Val_{S_j}(situação) =$ `viajando`.

Suponha que $Val_{S_i}(situação) =$ `pesquisando`. Neste caso, será executada a regra 3, a qual pode alterar o valor de *situação* de duas formas: (i) quando o livro é achado, o valor de *situação* muda para `achou`; (ii) quando o livro não é encontrado, o valor de *situação* muda para `não_achou`. Como a cada transição um novo livro é pesquisado, temos que em algum estado S_k, o caso (i) ou o caso (ii) se verificará e, portanto, $Val_{S_k}(situação) =$ `achou` ou $Val_{S_k}(situação) =$ `não_achou`. Logo, em S_k a única regra executável é a regra 4, a qual dá origem a um estado S_j, onde $Val_{S_j}(situação) =$ `final` ou $Val_{S_j}(situação) =$ `viajando`. □

Proposição 3 *Não havendo bloqueio, a pesquisa em todas as livrarias termina, e o sistema retorna ao ambiente* `Origem`.

Prova: Deve-se provar que se $Val_{S_0}(situação) =$ `inicial` e $Val_{S_0}(here) =$ `Origem`, então $Val_{S_j}(situação) =$ `final` e $Val_{S_j}(here) =$ `Origem`, para $j > i$.

No estado inicial S_0, por inspeção direta no texto da especificação, temos que Val_{S_0} *(situação)* = `inicial` e $Val_{S_0}(here) =$ `Origem`. Assim, executam-se sucessivamente as regras 1 e 2 e obtém-se um estado S_2, onde $Val_{S_2}(situação) =$ `pesquisando`. Pela proposição 1, o sistema alcançará um estado S_{j_1}, onde podem ocorrer dois casos:

1. $Val_{S_{j_1}}(situação) =$ `final`: Neste caso, temos também que $Val_{S_{j_1}}(here) =$ `Origem`, pois certamente terá sido disparada a regra 4.1 para atingir este estado. Logo, a proposição é satisfeita.
2. $Val_{S_{j_1}}(situação) =$ `viajando`: Neste caso, o estado S_{j_1} certamente foi obtido a partir do disparo da regra 4, a qual retira um dos ambientes da lista de livrarias. Em seguida, dispara-se a regra 2, obtendo-se um estado S_{i_2}, onde $Val_{S_{i_2}}(situação)$ = `pesquisando`. Assim, pela proposição 1, teremos novamente o caso 1 acima ou este próprio caso 2.

Portanto, por indução no comprimento da lista de livrarias, temos que em algum estado S_j, o caso 1 será escolhido e a proposição será satisfeita. □

7 Conclusões

Neste trabalho, apresentou-se um método para especificação formal de sistemas 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 sistemas móveis em ASM apresenta as seguintes vantagens:

- ASM é um método formal de fácil entendimento, exigindo pouca bagagem matemática de seus usuários.
- ASM já foi usada com sucesso para especificar uma variedade de sistemas computacionais.
- Especificações ASM são diretamente executáveis, permitindo que o usuário não apenas especifique, mas também simule o funcionamento do sistema real.

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

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

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

[5] 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.

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

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

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

[9] Lange, D.B. and Oshima, M. *Programming and Deploying Java Mobile Agents with Aglets.* Addison-Wesley, 1998.

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

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

[12] Milner, R. *Communicating and Mobile Systems: the Pi-Calculus.* Cambridge University Press, 1999.

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

[14] 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.

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