Um ambiente para especificação e verificação automática de aplicações baseadas em agentes móveis

Aline M. S. Andrade[1] **, Flávio M. Assis Silva**[1] **, Frederico J. R. Barboza**[1]
Rafael A. R. Oliveira[1]

[1]LaSiD - Laboratório de Sistemas Distribuídos
DCC - Departamento de Ciência da Computação
UFBA - Universidade Federal da Bahia
Av. Adhemar de Barros, S/N – Campus de Ondina
40170-110 Salvador, BA

{aline,fassis,fredjrb}@ufba.br, rafael_angelo@yahoo.com.br

Abstract. *In a previous work, we have proposed an extension of the Promela language which adds primitives for the specification of components of mobile agent based systems. Promela is the specification language used in the SPIN system, a model checker and simulation tool, very frequently used for the specification and verification of distributed protocols. This extension makes possible the specification of systems based on mobile agents by using primitives that directly model components of such systems as well as the use of SPIN as a model checker and simulation tool. In this paper we present the model upon which the proposed extension to Promela is based and a specification of a set of properties that this model satisfies. These properties were formally verified using SPIN. We demonstrate the applicability of the proposed extension by presenting an example of a specification that defines a simple mechanism for sending messages to mobile agents with migration transparency and by showing how a property of this specification could be formally proved with SPIN.*

Resumo. *Com o objetivo de se ter um ambiente para a especificação e verificação de sistemas baseados em agentes móveis, propusemos uma extensão da linguagem Promela que introduz primitivas para a especificação de componentes presentes neste sistemas. Promela é a linguagem de especificação utilizada no SPIN, um ambiente de verificação de modelos e simulação, bastante utilizado na análise de protocolos distribuídos. Esta extensão permite a especificação de sistemas baseados em agentes móveis utilizando-se primitivas que modelam diretamente componentes presentes em tais sistemas e a utilização do SPIN como ferramenta de verificação de modelos e simulação. Neste artigo apresentamos o modelo sobre o qual definimos a extensão de Promela e especificamos um conjunto de propriedades que este modelo satisfaz, mostrando como estas propriedades foram verificadas no SPIN. Demonstramos a aplicabilidade desta extensão apresentando um exemplo de uma especificação que define um mecanismo simples de entrega de mensagens para agentes móveis com transparência de migração e mostrando como uma propriedade desta especificação pode ser formalmente provada sobre SPIN.*

1. Introdução

Nos últimos anos o uso de agentes móveis tem sido proposto para o desenvolvimento de sistemas/aplicações em diversas áreas, como, por exemplo, gerenciamento de redes, comércio eletrônico e redes ativas [Fuggetta et al., 1998]. Um agente móvel é um componente de software capaz de migrar autonomamente entre nós de um sistema distribuído. Sua utilização no desenvolvimento de tais sistemas é proposta devido a benefícios que podem trazer na implementação de sistemas distribuídos, tais como melhoria do uso dos recursos de comunicação, melhor uso dos recursos computacionais, suporte à extensibilidade de sistemas e suporte à operação desconectada.

No entanto, garantir a correção de sistemas baseados em agentes móveis não é uma tarefa trivial. Tais sistemas são inerentemente distribuídos e concorrentes. Adicionalmente, possuem componentes que podem migrar autonomamente no ambiente distribuído, completamente independentes uns dos outros. Em ambientes abertos, como a Internet, fatores adicionais aumentam a complexidade de se garantir a correção de sistemas baseados em agentes móveis, como a possibilidade de agentes estarem se movendo em um ambiente formado por diversas organizações autônomas e sujeito a falhas.

O uso de ferramentas baseadas em métodos formais tem sido atualmente uma das técnicas utilizadas para se garantir a correção de sistemas seqüenciais e concorrentes e sua aplicação tem se estendido também aos sistemas baseados em agentes móveis [Unyapoth, 2001, de Nicola et al., 1998, Wojciechowski e Sewell, 1999, Dotti e Ribeiro, 2000, Fournet et al., 1996]. Estes métodos permitem a representação de sistemas através de um modelo matemático com sintaxe e semântica bem definidas. Portanto, podem ser utilizados como uma ferramenta para especificar sistemas de forma precisa e conseqüentemente permitir a verificação formal de propriedades.

Um dos paradigmas para verificação formal que vem sendo utilizado com sucesso na prática é o método de verificação de modelos. A verificação de modelos se baseia normalmente em uma representação do sistema em um modelo de estados (sistema de transição) e de propriedades a serem verificadas em relação a este sistema especificadas em lógica temporal. Com o uso de uma ferramenta automática que analisa os estados do sistema, pode-se verificar se as propriedades especificadas são satisfeitas ou não.

O SPIN [Holzmann, 1991] é um ambiente que fornece ferramentas automáticas para simulação e verificação de modelos, que tem sido bastante usado no desenvolvimento de aplicações distribuídas e projetos de protocolos. SPIN permite a criação dinâmica de processos concorrentes, que podem se comunicar via canais de mensagens síncronos ou assíncronos.

A linguagem de especificação utilizada pelo SPIN, Promela, não possui primitivas para a especificação de mobilidade, ou seja, primitivas que expressem explicitamente componentes móveis (agentes) e seus movimentos. Propusemos em [Andrade et al., 2002] uma extensão da linguagem Promela para possibilitar a modelagem explícita de sistemas baseados nestes componentes. Em particular, foram introduzidas na linguagem primitivas para a especificação de agentes móveis, do ambiente onde executam (*agência*), de suas ações de movimento, da comunicação entre agentes e do comportamento dos mecanismos de comunicação em relação a falhas. Utilizando-se um tradutor que mapeia estas primitivas em primitivas originais de Promela, SPIN pode ser usado para

verificar e simular aplicações baseadas em agentes móveis. O benefício desta extensão é que podemos utilizar um ambiente com ferramentas para simulação e verificação, bastante usado na prática, para a análise de aplicações móveis, tendo o especificador primitivas que modelam diretamente conceitos da sua área de aplicação (conceitos de um ambiente de agentes móveis). Além disso, a linguagem Promela é uma linguagem bastante simples, com uma estrutura semelhante à de uma linguagem de programação. Entendemos que esta característica facilita a produção de especificações por pessoas que não possuem experiência prévia com técnicas de especificação formal, mas que têm conhecimento de linguagens de programação de uso geral.

Neste artigo nos concentramos no aspecto de verificação de sistemas sobre a extensão proposta em [Andrade et al., 2002]. Inicialmente apresentamos um conjunto de propriedades que a nossa extensão de Promela garante. Depois apresentamos um exemplo de especificação em que um mecanismo de comunicação entre agentes móveis simples é definido e mostramos como este mecanismo pode ser verificado sobre nosso modelo. A definição de um conjunto de propriedades que nosso modelo fornece é fundamental para a verificação incremental de sistemas. O exemplo que apresentamos, embora mantido simples por questão de espaço do artigo, representa um problema básico no desenvolvimento de sistemas de agentes móveis. Diferentes soluções para este problema foram propostas. Estas soluções podem ser especificadas e verificadas sobre nosso sistema. Em particular, estaremos usando este sistema para a especificação e verificação de protocolos de tolerância a falhas e de comunicação confiável para agentes móveis [Araújo Macêdo e Assis Silva, 2002].

Este artigo está estruturado da seguinte maneira. Na seção 2 apresentamos uma descrição genérica do modelo adotado de agentes móveis e de seu ambiente. Na seção 3 descrevemos a extensão feita à linguagem Promela com as primitivas para modelar aplicações baseadas em agentes móveis. Na seção 4 apresentamos um exemplo de uso desta extensão para a especificação de um mecanismo de comunicação entre agentes com transparência de migração. Na seção 5 apresentamos as principais propriedades que nosso modelo garante. Na seção 6 mostramos a verificação do exemplo proposto. Na seção 7 discutimos trabalhos relacionados. Finalmente, na seção 8 apresentamos as conclusões.

2. Modelo do Ambiente de Agentes Móveis

Um *agente móvel* é um elemento de software capaz de autonomamente migrar entre nós de um sistema distribuído. Mais especificamente, um agente migra de uma *agência* a outra. De um ponto de vista conceitual, uma agência representa um lugar lógico no sistema distribuído. De um ponto de vista funcional, uma agência representa a funcionalidade necessária em um nó para suportar o ciclo de vida de agentes (criação, execução, movimento, etc.). Quando um agente migra, sua execução é suspendida na agência original e o agente é transportado (código, dados e estado de execução) para uma outra agência no ambiente distribuído, onde a execução é retomada. Durante sua execução, um agente móvel pode criar um ou mais agentes móveis, chamados de seus *agentes filhos*. A figura 1 ilustra três agentes, $ag1$, $ag2$ e $ag3$, e cinco agências, *agência1, agência2, agência3, agência4* e *agência5*. Na figura, o agente $ag1$ moveu-se de *agência1* para a *agência2* e, nesta agência, criou dois agentes filhos, $ag2$ e $ag3$. Cada um deles migrou para uma agência distinta.

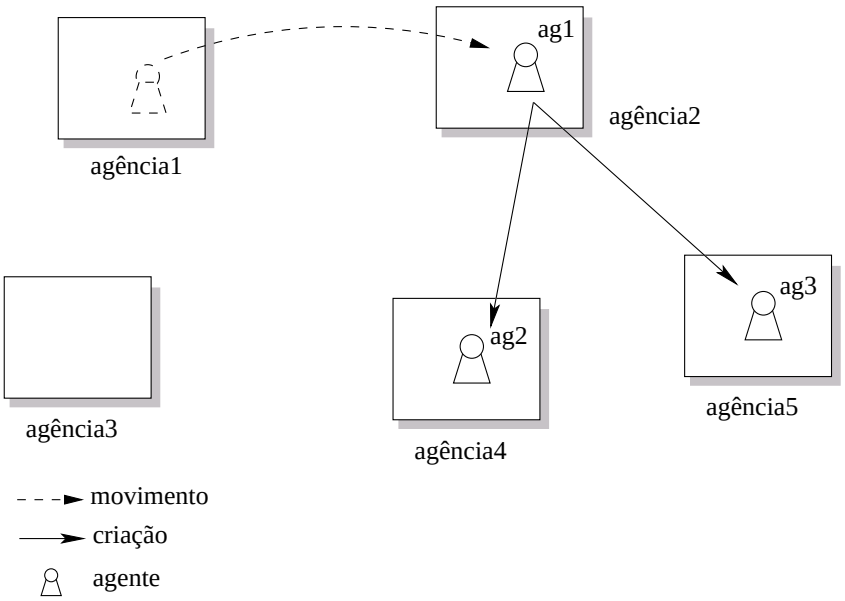

Figura 1: Modelo do ambiente de agentes móveis

Em aplicações baseadas em agentes móveis, as agências, além das funções básicas de manter o ciclo de vida dos agentes (como receber agentes, instanciar agentes e enviá-los para outras agências), podem também executar funções específicas da aplicação. Por exemplo, manter algum repositório de informação que os agentes queiram acessar naquela agência. Em nosso modelo, cada agência executa dois tipos de atividades: *atividades básicas* e *atividades específicas*. As atividades básicas representam atividades que toda agência realiza, independentemente de aplicação. O comportamento específico de uma agência corresponde a ações dependentes de aplicação, a serem executadas na ocorrência de determinados eventos. Por exemplo, uma agência em uma determinada aplicação pode necessitar ter um comportamento específico sempre que um agente chega na agência.

Agentes e agências comunicam-se entre si trocando mensagens através de canais. Alguns sistemas de agentes móveis existentes atualmente provêm formas de comunicação entre agentes que suportam entrega de mensagens com *transparência de migração*, ou seja, uma mensagem enviada a um agente é entregue a ele independentemente de sua migração (por exemplo, [ObjectSpace Inc., 1997]). Outros sistemas só mantêm o canal de comunicação entre agentes enquanto eles não migram (por exemplo, [Baumann et al., 1997]). No nosso modelo, como em *Nomadic Pict* [Unyapoth, 2001], o emissor de uma mensagem deve especificar a agência onde o recebedor estará. Fornecendo esta forma mais básica de comunicação, o modelo pode ser usado para se especificar diferentes formas de se entregar mensagens com transparência de migração.

A comunicação entre agentes e entre agências é assíncrona, ou seja, não são conhecidos limites de tempo máximos para que uma mensagem seja transmitida de um agente a outro ou de uma agência a outra. Além de se comunicar com outros agentes, um agente pode se comunicar também com a agência onde se encontra. A comunicação entre um agente e uma agência é, no entanto, síncrona. Não há limites conhecidos também para o tempo de processamento relativo de agentes e agências.

Em relação a confiabilidade, dois tipos de canais são considerados: canais con-

fiáveis, onde mensagens são eventualmente entregues em seus destinos sem duplicação, perda nem corrompimento; e canais não confiáveis, onde mensagens podem ser perdidas, mas não duplicadas nem corrompidas. Agentes e agências não falham.

3. Extensão de Promela para a Representação de Componentes de um Ambiente de Agentes Móveis

Nesta seção descrevemos a extensão de Promela proposta em [Andrade et al., 2002] para a especificação de sistemas baseados em agentes móveis. Nesta extensão foram criadas primitivas para a especificação de agentes móveis, de agências, de ações de movimento de agentes, da comunicação entre agentes e do comportamento dos canais de comunicação em relação a falhas.

Em uma especificação de sistema baseado em agentes móveis poderão ser usadas as primitivas introduzidas em [Andrade et al., 2002] e primitivas originais de Promela. A especificação é então traduzida para uma outra, em que as primitivas introduzidas para representar componentes de sistemas de agentes móveis são mapeadas em primitivas e componentes originalmente existentes em Promela. Esta especificação resultante pode ser utilizada no SPIN para se fazer verificação e simulação.

Agências

Uma agência é especificada utilizando-se a primitiva *agency*. A sintaxe para a especificação de uma agência é ilustrada abaixo:

```
agency <nome da agencia>{
  onAgentArrival(AgencyMsg e){
   <comandos>
  }
  onAgentLeave(AgencyMsg e){
   <comandos>
  }
  onMessageDiscard(AgencyMsg e){
   <comandos>
  }
  onAgencyStart{
   <comandos>
  }
}
```

Ao se especificar uma agência, indicamos seu nome (`<nome da agencia>`) e seu corpo de comandos. Em seu corpo de comandos podemos especificar ações a serem executadas pela agência na ocorrência de quatro eventos:

- chegada de um agente: as ações a serem executadas quando este evento ocorre são especificadas no bloco de instruções *onAgentArrival*;
- movimento de um agente para outra agência: as ações a serem executadas quando este evento ocorre são especificadas no bloco de instruções *onAgentLeave*;
- mensagem descartada: este evento ocorre quando a agência possui uma mensagem para ser entregue a um agente que não se encontra nesta agência. As ações a

serem tomadas na ocorrência deste evento são especificadas no bloco de instruções *onMessageDiscard*;

- início de execução de uma agência: as ações a serem tomadas no início da execução de uma agência são especificadas no bloco *onAgencyStart*.

Estes eventos representam situações típicas onde sistemas sendo especificados podem ter comportamentos próprios a serem determinados.

Na ocorrência dos três primeiros eventos, podemos operar sobre uma variável do tipo *AgencyMsg*. *AgencyMsg* é um tipo embutido da extensão e contém informações sobre o evento específico que ocorreu. Por exemplo, no caso do evento *onAgentArrival*, contém informações sobre o agente que chegou e de qual agência ele vem.

No bloco *onAgencyStart*, o comando *start* pode ser utilizado para disparar a execução de agentes na agência. Este comando tem a sintaxe mostrada abaixo:

```
start <nome do agente>
```

No comando acima, o nome do agente a ser criado é passado como parâmetro (a sintaxe para a especificação de agentes é mostrada logo a seguir no texto).

As ações de tratamento dos eventos correspondem ao comportamento específico da agência. O comportamento genérico da agência é incorporado a cada agência de maneira implícita pelo nosso sistema e corresponde a funcionalidades para implementar o transporte de agentes e o mecanismo básico de comunicação entre agentes.

Agentes

Uma especificação de um agente possui a seguinte estrutura geral:

```
agent <nome do agente>{
  <comandos>
}
```

Os comandos presentes na parte indicada com `<comandos>` definirão o comportamento do agente. Além das instruções de Promela, três primitivas especiais podem ser usadas, *move*, *send* e *receive*. A instrução *move*, quando executada, resulta na migração do agente de uma agência para outra. Um comando *move* possui a seguinte estrutura:

```
move (<nome da agencia destino>, <resultado da operacao>)
```

Durante uma operação de movimento, dependendo da propriedade da rede de comunicação em relação a falhas, a migração pode ocorrer com sucesso ou não. Se a migração ocorrer com sucesso, a execução do agente continuará a ser realizada na agência destino. Se o movimento for mal sucedido, o agente permanecerá na sua agência de origem (o resultado da operação é informado no parâmetro `<resultado da operacao>`). As primitivas *send* e *receive* serão descritas mais abaixo.

Comunicação e a Rede de Troca de Mensagens

Cada especificação possui, implicitamente, um componente que representa a rede de comunicação entre agentes. Esta rede poderá se comportar de forma confiável ou não. No caso de uma rede confiável, uma mensagem enviada será eventualmente entregue e

uma migração de um agente sempre termina com sucesso. No caso da rede não confiável, poderá haver omissão de mensagens e a operação de movimento de um agente poderá terminar sem sucesso. Ao se fazer uma especificação, informa-se o tipo de comportamento a ser adotado pela rede, utilizando-se as instruções:

```
reliable network
```

ou

```
unreliable network
```

Estas instruções devem aparecer como a primeira instrução do arquivo com a especificação do sistema. Elas afetam os canais de comunicação entre agentes. A comunicação entre um agente e uma agência onde o agente se encontra é sempre confiável. Canais confiáveis se comportam de forma FIFO (*first-in, first-out*), ou seja, mensagens enviadas antes são consumidas antes.

A troca de mensagens entre agentes é especificada com a primitiva *send*:

```
send (<agente>, <agencia>, <mensagem>)
```

A mensagem `<mensagem>` é enviada ao agente `<agente>` na agência `<agencia>`. O envio da mensagem é feito sem transparência de localização, ou seja, o agente emissor deve indicar a agência onde o agente recebedor deve estar. Quando a mensagem enviada chegar na agência destino, a agência recebe a mensagem e a entrega ao agente. Se o agente não estiver na agência quando esta for entregar a mensagem, um evento *onMessageDiscard* é gerado. Como descrito acima, ações podem ser especificadas para serem executadas quando um evento deste tipo for gerado (por exemplo, para encaminhar a mensagem a algum componente que tentará enviar a mensagem para um novo local onde o agente deve estar).

Para ler uma mensagem da rede, o agente deve fazer uso da primitiva *receive*:

```
receive (<tipo>, <emissor>, <localizacao>, <mensagem>)
```

Nesta primitiva o parâmetro `<tipo>` é um parâmetro de entrada. Os demais parâmetros são de saída. O parâmetro `<tipo>` indica o tipo da mensagem a ser lida. O parâmetro `<emissor>` conterá o nome do agente que enviou a mensagem. O parâmetro `<localizacao>` indica a agência onde o agente emissor estava localizado no momento em que enviou a mensagem. O parâmetro `<mensagem>` conterá a mensagem em si.

4. Exemplo

Nesta seção apresentamos uma especificação de um mecanismo de entrega de mensagens a agentes móveis com transparência de migração. Na primitiva de envio de mensagem apresentada na seção anterior, é necessário especificar uma agência para onde a mensagem será enviada. O exemplo desta seção ilustra o uso das extensões de Promela para especificar um mecanismo de envio de mensagens que elimina esta necessidade. Este exemplo, embora simples, representa um problema real na construção de sistemas de agentes móveis. Para este problema várias soluções foram propostas e o sistema apresentado neste artigo representa uma abordagem para comparar e validar estas propostas.

As figuras 2 e 3 apresentam uma parte da especificação. No exemplo estão especificados três agentes (*Ag0, Ag1 e Ag2*) (figura 2) e quatro agências (figura 3). Um dos agentes, o *Ag0*, não se moverá e desempenhará a função de um servidor de entrega de mensagens para agentes. O agente *Ag2* também não se move e será um agente que enviará uma mensagem para o *Ag1*. O agente *Ag1*, por sua vez, migra entre agências. O agente *Ag0* executará na agência *site4* (veja figura 3, linha 22, onde este agente é criado). O agente *Ag2* executará na agência *site2* (veja figura 3, linha 10, onde este agente é criado). O agente *Ag1* será criado na agência *site1* (veja figura 3, linha 3).

O agente *Ag0* mantém a informação sobre a localização corrente de *Ag1*. Para isto, um protocolo é executado entre os agentes *Ag0* e *Ag1*. O agente *Ag0* pode receber os seguintes tipos de mensagens: WANT_MOVE, MOVED e M_MSG. Uma mensagem do tipo WANT_MOVE é enviada pelo agente *Ag1* quando ele deseja migrar. Ao receber uma mensagem deste tipo, o agente *Ag0* envia uma mensagem de *acknowledgement* (ACK_W) para o *Ag1* (figura 2, linha 10). Após ter-se movido, o agente *Ag1* envia uma mensagem do tipo MOVED para o agente *Ag0*, informando a nova agência onde se encontra. Quando *Ag0* recebe uma mensagem deste tipo, ele verifica se há alguma mensagem armazenada em um *buffer* (variável *bf*). Caso haja, ele a envia a *Ag1* (figura 2, linha 13). Quando *Ag0* recebe uma mensagem para ser entregue a *Ag1*, ele verifica se *Ag1* está se movendo (figura 2, linha 18). Se estiver (variável *pos* é igual a -1), ele armazena a mensagem em um *buffer*. Se não, ele a envia a *Ag1*.

O *Ag1* é criado na agência *site1* e depois se move para a agência *site2*, *site3* e *site4*, nesta ordem. Quando ele vai se mover, ele envia uma mensagem do tipo WANT_MOVE para *Ag0* (a mensagem é montada nas linhas 35 a 41 da fig. 2 e enviada na linha 30 da mesma figura). Nesta mensagem é indicada a agência para onde quer se mover. Após enviar uma mensagem do tipo WANT_MOVE, *Ag1* espera por uma mensagem do tipo ACK_W de *Ag0*. Quando esta mensagem chega, ele envia uma mensagem do tipo MOVED para *Ag0* informando o término do movimento (agência *site4*) (fig. 2, linha 35).

A especificação de *Ag2* apenas envia uma mensagem a *Ag1* (fig. 2, linhas 47 a 51). A especificação das agências (fig. 3) consiste apenas na especificação da criação dos agentes: *Ag1* na agência *site1* (linha 3), *Ag2* na agência *site2* (linha 10) e *Ag0* na agência *site4* (linha 22). A agência *site3* é especificada apenas como uma agência para onde o agente *Ag1* irá se mover.

5. Propriedades do Modelo

Na descrição do modelo apresentado na seção 2, uma série de propriedades foi atribuída ao mesmo de maneira informal. Para a formalização e verificação destas propriedades, definimos fórmulas em Lógica Temporal Linear (LTL) especificadas sobre estados de processos Promela. Por questão de espaço, apresentamos nesta seção apenas as fórmulas LTL, mostrando de maneira informal o significado das proposições (fazendo-se uma abstração sobre os estados dos processos envolvidos).

As propriedades definidas foram provadas como corretas utilizando-se o verificador de modelos do SPIN. A garantia formal destas propriedades torna a extensão de Promela aplicável, uma vez que especificações de sistemas baseados em agentes móveis poderão ser feitas considerando-as. A especificação das propriedades garantidas por nosso

```
01 reliable network;
02 #define NUM_AGENCY 4
03 mtype = {M_MSG, ACK_W, WANT_MOVE, MOVED};
04 ...
05 agent Ag0 {
06     ...
07     do
08         ::receive(type, sender, site, msg)->
09             if
10                 ::(msg.info==WANT_MOVE)->pos=-1;answer.info=ACK_W;send(sender,site,answer);
11                 ::(msg.info==MOVED)->pos=site;
12                     if
13                         ::(bf.destAg!=-1)->send(bf.destAg, site, bf);bf.destAg=-1;
14                         ::else->skip;
15                     fi;
16                 ::(msg.info==M_MSG)->
17                     if
18                         ::(pos>=0)->send(msg.destAg, pos, msg);
19                         ::else->copyPacket(msg,bf);
20                     fi;
21             fi;
22     od;
23 };
24
25 agent Ag1{
26     ...
27     byte canMove=1;
28     ...
29     do
30         ::(canMove==1)->canMove=0;send($Ag0$, $$site4$$, ask);
31         ::receive(type, sender, site, msg)->
32             if
33                 ::(msg.info==M_MSG)->msgRecv=1;
34                 ::(msg.info==ACK_W)->move(nextSite, resultado);
35                     ask.info=MOVED; send($Ag0$, $$site4$$, ask);
36                     ask.info = WANT_MOVE; nextSite=nextSite+1;
37                     if
38                         ::(nextSite >= NUM_AGENCY)->canMove=0;
39                         ::else->canMove=1;
40                     fi;
41                     ask.site=nextSite;
42             fi;
43     od;
44 };
45
46 agent Ag2{
47     msg.info = M_MSG;
48     msg.destAg = $Ag1$;
49     ...
50     msgSent=1;
51     send($Ag0$, $$site4$$, msg);
52 };
```

Figura 2: Exemplo do Uso da Extensão de Promela

```
01 agency site1{                    14 agency site3{
02     onAgencyStart{               15     onAgencyStart{
03       start Ag1;                 16       ...
04       ...                        17     }
05     }                            18 }
06 }                                19
07                                  20 agency site4{
08 agency site2{                    21     onAgencyStart{
09     onAgencyStart{               22       start Ag0;
10       start Ag2;                 23       ...
11       ...                        24     }
12     }                            25 }
13 }
```

Figura 3: Exemplo do Uso da Extensão de Promela

modelo também possibilita a verificação incremental de sistemas. Ou seja, especificações de sistemas baseados em agentes móveis feitas a partir de nossa extensão de Promela podem ser verificadas de forma automática ou não a partir do conhecimento destas propriedades. A necessidade de especificação não automática pode ocorrer em situações onde ocorra, por exemplo, explosão de estados ou a existência de infinitos estados.

Nas especificações abaixo, os operadores de LTL $\Diamond$P e $\Box$P significam, respectivamente, "em algum tempo futuro ocorrerá P" e "sempre ocorrerá P".

Prop 1 (Consistência no envio de mensagens) *Uma mensagem somente é recebida se tiver sido enviada. Em LTL, expressamos esta propriedade com a fórmula:*

$$\Box(received \Rightarrow sent)$$

onde sent é uma proposição que se torna verdadeira quando o agente emissor enviar uma mensagem. A proposição received se torna verdadeira quando a mensagem é recebida na agência destino.

Prop 2 (Entrega eventual de mensagem) *Se a rede é declarada como confiável (*`reliable network`*), toda mensagem enviada por um agente chegará, em algum momento futuro, à agência destino. Em LTL, expressamos esta propriedade com a fórmula:*

$$\Box(sent \Rightarrow \Diamond received)$$

onde sent e received possuem a mesma semântica que na propriedade anterior.

Prop 3 (Não duplicação de mensagens) *Se a rede é declarada como confiável (*`reliable network`*), toda mensagem enviada por um agente chegará apenas uma vez à agência destino. Em LTL, a propriedade é descrita pela fórmula:*

$$\Box((sent \Rightarrow \Diamond received_1) \wedge \neg received_2)$$

onde $sent$ é uma proposição que se torna verdadeira quando o agente emissor envia a mensagem. A proposição $received_1$ se torna verdadeira quando a agência destino recebe a mensagem pela primeira vez. A proposição $received_2$ é verdadeira se a agência destino receber a mensagem pela segunda vez.

Prop 4 (Canais FIFO) *Se a rede é especificada como confiável (*`reliable network`*), se duas mensagens são enviadas seqüencialmente e as duas chegam à agência destino, a primeira sempre chegará antes que a segunda. Em LTL, descrevemos a propriedade com o uso da fórmula:*

$$\Box((sent_2 \Rightarrow sent_1) \Rightarrow (received_2 \Rightarrow received_1))$$

onde as proposições $sent_1$ e $sent_2$ se tornam verdadeiras, respectivamente, quando a primeira e a segunda mensagem são enviadas. As proposições $received_1$ e $received_2$ se tornam verdadeiras, respectivamente, quando a primeira e a segunda mensagem são recebidas. Esta propriedade é garantida, assumindo-se a propriedade 1.

Prop 5 (Movimento consistente) *Um agente não se move para uma agência se não tiver executado uma instrução de movimento para aquela agência. Em LTL escrevemos:*

$$\Box(newHost \Rightarrow moved)$$

onde $moved$ é uma proposição que é interpretada como verdadeira quando o agente executa a operação de se mover e $newHost$ é uma proposição interpretada como verdadeira quando o ambiente de execução do agente se torna a agência destino.

Prop 6 (Movimento confiável) *Se a rede é declarada como confiável (*`reliable network`*), toda operação de movimento termina com sucesso e o agente continuará a executar, em algum momento futuro, na agência destino. Em LTL, usamos a fórmula:*

$$\Box((moved \Rightarrow \Diamond(newHost \wedge executed)) \wedge (executed \Rightarrow newHost))$$

onde $moved$ e $newHost$ assumem a mesma interpretação que na propriedade anterior e $executed$ é uma proposição interpretada como verdadeira quando o agente volta a executar após o movimento.

Prop 7 (Indicação de falha durante movimento) *Se a rede é declarada como não confiável (*`unreliable network`*) e uma operação de movimento falhar, o agente continuará a executar na mesma agência. Em LTL, descrevemos a propriedade com a fórmula:*

$$\Box((moved \wedge failed) \Rightarrow (executed \wedge \neg newHost))$$

com as proposições moved, executed e newHost assumindo semânticas idênticas àquelas descritas nas propriedades 5 e 6. A proposição failed é verdadeira se o agente que estiver se movendo falhar.

As propriedades acima foram verificadas assumindo-se a hipótese de *justiça no escalonamento de processos* (*fairness*). Ou seja, sempre vale que todos os processos ativos envolvidos na especificação serão em algum ponto no futuro escalonados para executar. Sem esta hipótese, não se pode garantir algumas das propriedades listadas acima. Por exemplo, se houver em uma especificação um agente que avance indefinidamente, sem trocar mensagens com outros agentes, a propriedade de entrega eventual de mensagens não será garantida entre outro par de agentes. Isto acontece porque existirão caminhos computacionais em que agências, agentes e a rede de comunicação não terão a oportunidade de prosseguir na sua execução. O modelo do SPIN não garante justiça de escalonamento. Portanto, ao se fazer uma especificação sobre nossa extensão de Promela, o especificador deve se certificar de que a especificação de nenhum de seus processos (agentes) possa impedir a execução de outros processos (agentes).

6. A Verificação do Exemplo

Nesta seção exemplificamos o processo de verificação de uma especificação feita sobre a extensão de Promela, através da validação, para o exemplo descrito na seção 4, da propriedade de entrega eventual de mensagens *entre agentes*. Na seção 5 foi mostrada que o sistema apresentado neste artigo garante que uma mensagem enviada por um agente eventualmente chega, em uma rede confiável, à agência destino. O que se quer validar é a entrega eventual de uma mensagem de um agente a outro (ou seja, *fim-a-fim*) no exemplo especificado na seção 4.

Para demonstrar a propriedade, consideraremos os agentes *Ag2*, emissor da mensagem, e *Ag1*, agente para o qual a mensagem se destina. Uma sentença em LTL que expressa a propriedade que queremos mostrar é:

$$\Box(sent\_by\_agent \Rightarrow \Diamond received\_by\_agent)$$

onde a proposição *sent_by_agent* é verdadeira quando a mensagem é enviada por *Ag2* e *received_by_agent* é verdadeira quando a mensagem é recebida por *Ag1*. Na especificação, a propriedade *sent_by_agent* ficará verdadeira quando a variável *msgSent* receber o valor 1 (fig. 2, linha 50). A proposição *received_by_agent* ficará verdadeira quando a variável *msgRecv* receber o valor 1 (fig. 2, linha 33).

Para verificar a propriedade, primeiro a sentença LTL acima foi fornecida ao verificador de modelos do SPIN (utilizando sua notação específica). Ao se considerar uma rede confiável, a propriedade foi avaliada como verdadeira, ou seja, uma mensagem enviada por *Ag2* eventualmente é recebida por *Ag1*, apesar de suas movimentações.

Este comportamento é esperado em virtude de o movimento do *Ag2* ser monitorado e controlado por *Ag0* através de um protocolo composto pelas mensagens WANT_MOVE, ACK e MOVED e de o agente *Ag1* interromper seu movimento em um determinado instante (agência *site4*). Para *Ag0* é sempre possível determinar a localização correta de *Ag1*.

A seguir, a especificação foi alterada para executar sobre um ambiente de rede não confiável. Para isto bastou alterar a primeira linha da especificação (trocou-se a linha *reliable network* para *unreliable network*). Neste caso a possibilidade de perda de mensagens faz com que não se possa garantir que a mensagem enviada chegue a seu destino.

7. Trabalhos Relacionados

Muitos formalismos foram propostos para a especificação de sistemas móveis, como cálculo-π [Milner, 1999], *Mobile Ambients* [Cardelli e Gordon, 1998], *Distributed Join Calculus* [Fournet et al., 1996] e *Nomadic π-calculus* [Unyapoth, 2001]. Os formalismos diferem na maneira como modelam mobilidade ou na habilidade de explicitamente modelarem aspectos específicos do desenvolvimento de aplicações/protocolos móveis. Linguagens de programação como *Pict* [Pierce e Turner, 1997] e *Nomadic Pict* [Wojciechowski e Sewell, 1999] são baseadas em cálculo-π e *Nomadic π-calculus*, respectivamente. De nosso conhecimento, apenas uma ferramenta realiza verificação de modelos a partir de formalismos para sistemas móveis. Esta ferramenta, chamada *Mobility*

Workbench [Victor, 1995], utiliza cálculo-π como linguagem de especificação. Adicionalmente, em [Song e Compton, 2003] é descrito um mapeamento de cálculo-π para Promela. Com este mapeamento, SPIN pode ser usado para se fazer verificação de modelos de sistemas especificados em cálculo-π.

A extensão que propomos de Promela utiliza noções explícitas de agentes e localizações (como em *Mobile Ambients, Nomadic π-Calculus* e *Distributed Join Calculus*), e a localização de um agente é especificada através de um canal entre o agente e sua agência (de modo análogo a cálculo-π). De forma similar a *Join Calculus*, a nossa extensão possui também uma semântica simples de falhas. Ao contrário de *Mobile Ambients*, aspectos de segurança não são tratados. De maneira semelhante a *Nomadic Pict*, nós introduzimos primitivas de comunicação dependentes de localização com as quais infra-estruturas para comunicação independente de localização podem ser expressas. Em comparação com *Mobility Workbench* e [Song e Compton, 2003], ambos utilizam cálculo-π como linguagem de especificação. Consideramos que a ferramenta que propomos se baseia em uma linguagem de estrutura semelhante à de linguagens de programação de uso freqüente. Isto facilita a geração de especificações e torna mais simples a incorporação de propriedades ao modelo, como tratamento de falhas, por exemplo.

O trabalho apresentado neste artigo foi desenvolvido no contexto dos projetos ForMOS, IQ-Mobile e ARGO. No contexto dos dois primeiros projetos foi proposto também um ambiente para especificação, simulação, verificação e geração de código de sistemas móveis baseado em Gramáticas de Grafos Baseadas em Objetos (OBGG) [Dotti e Ribeiro, 2000]. Com o desenvolvimento das duas abordagens, espera-se chegar a uma metodologia para o desenvolvimento de aplicações móveis corretas.

8. Conclusão

Neste trabalho, estendemos a linguagem Promela com primitivas para modelar sistemas de agentes móveis. Criamos um modelo de mobilidade, onde representamos explicitamente o conceito de agente, agência, comunicação entre agentes e movimento de agentes. Implementamos este modelo sobre Promela, e, com esta abstração, permitimos que o desenvolvedor de um sistema baseado em agentes móveis utilize o SPIN para especificar, verificar e simular o seu sistema utilizando primitivas que modelam diretamente componentes presentes em sistemas baseados em agentes móveis. Para o modelo de agentes móveis apresentado foi definido um conjunto de propriedades que foram verificadas formalmente na extensão criada, utilizando o próprio SPIN. Portanto, geramos uma extensão de Promela correta em relação às propriedades do modelo. Além disso, o modelo fornece um conjunto de funcionalidades básicas relativas a agências e permite que o usuário especifique propriedades específicas da sua aplicação.

A possibilidade de se poder especificar protocolos de sistemas de agentes móveis e de poder verificá-los sobre o sistema descrito neste artigo, com um conjunto de propriedades bem definido, torna-o uma ferramenta de auxílio no projeto de tais protocolos. Pretendemos utilizar este sistema para a especificação e validação de protocolos de confiabilidade para sistemas de agentes móveis (grupos móveis [Araújo Macêdo e Assis Silva, 2002]).

Estamos, atualmente, trabalhando em uma expansão do modelo para permitir a

especificação de sistemas na presença de outros tipos de falhas. Também está em andamento uma ferramenta para geração automática de código Java a partir de especificações em Promela estendida. O nosso objetivo é criar um ambiente para especificação, verificação e geração de código de sistemas de agentes móveis na presença de falhas de maneira a produzir sistemas com alto grau de confiabilidade.

Referências

Andrade, A. M., Assis Silva, F. M., Barboza, F. J. (2002). Extensão da linguagem promela para especificação de sistemas baseados em agentes móveis. In *Anais do IV Workshop de Comunicação Sem Fio e Computação Móvel - WCSF2002*, São Paulo.

Araújo Macêdo, R.J., Assis Silva, F.M. (2002). Coordination of mobile processes with mobile groups. In *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks - DSN2002*, pp.177–186, Washington, USA.

Baumann, J., Hohl, F., Rothermel, K., Strasser, M. (1997). Mole - concepts of a mobile agent system. Technical Report 1997/15, Universität Stuttgart, Fakultät Informatik.

Cardelli, L., Gordon, A. D. (1998). Mobile ambients. *Lecture Notes in Computer Science*, 1378.

de Nicola, R., Ferrari, G. L., Pugliese, R. (1998). Klaim: A kernel language for agents interaction and mobility. *IEEE Transactions on Software Engineering*, 24(5):315–330.

Dotti, F. L., Ribeiro, L. (2000). Code mobility in open systems: A formal approach. In *Parallel and Distributed Processing Techniques and Applications*.

Fournet, C., Gonthier, G., Levy, J.-J., Maranget, L., Rémy, D. (1996). A calculus of mobile agents. *Lecture Notes in Computer Science*, 1119:406–421.

Fuggetta, A., Picco, G., Vigna, G. (1998). Understanding code mobility. *IEEE Transactions on Software Engineering*, 24(5):342–361.

Holzmann, G. (1991). *Design and Validation of Computer Protocols*. Prentice Hall.

Milner, R. (1999). *Communicating and Mobile Systems: the π-calculus*. Cambridge University Press.

ObjectSpace Inc. (1997). Voyager technical overview.

Pierce, B.C., Turner, D.N. (1997). Pict: A programming language based on the pi-calculus. Technical Report 476, CSCI, Indiana University.

Song, H., Compton, K. J. (2003). Verifying π-calculus processes by promela translation. Technical Report CSE-TR-472-03, University of Michigan.

Unyapoth, A. (2001). *Nomadic π-Calculi: Expressing and Verifying Communication Infrastructure for Mobile Computation*. PhD thesis, pembroke college, University of Cambridge.

Victor, B. (1995). The mobility workbench user's guide: Polyadic version 3.122. Technical Report. Uppsala Universitet.

Wojciechowski, P. T., Sewell, P. (1999). Nomadic pict: Language and infrastructure design for mobile agents. In *Proceedings of ASA/MA'99 - First International Symposium on Agent Systems and Applications/Third International Symposium on Mobile Agents*.