Unificando os Modelos de Computação Móvel e Componentes de Software via Máquinas de Estado Interativas

Marcelo de Almeida Maia
marcmaia@iceb.ufop.br
Universidade Federal de Ouro Preto - DECOM
Campus Morro do Cruzeiro - Ouro Preto MG - 35400-000

Roberto da Silva Bigonha
bigonha@dcc.ufmg.br
Universidade Federal de Minas Gerais - DCC
Campus da Pampulha - Belo Horizonte MG - 30161-970

Resumo

Neste trabalho apresentamos as Máquinas de Estados Abstratas Interativas [10] [7] como uma alternativa para formalização de sistemas de computação móvel e de componentes de software. O resultado é que a simples idéia de troca de mensagens e reconfiguração dinâmica da topologia de comunicação fornece uma linguagem suficientemente expressiva para estas classes de problemas.

Abstract

In this work we show how Interactive Abstract State Machines [10] [7] can constitute an alternative for the formalization of mobile computing systems and software components. The result is that the simple idea of message passing combined with dynamic reconfiguration of the communication topology provides a sufficiently expressive language for these classes of problems.

1 Introdução

Segundo Peter Wegner[13], um problema central na especificação de sistemas de computação é a interação entre os componentes do sistema. Segundo ele, algoritmos modelam funções, enquanto sistemas interativos modelam os programas do mundo real. Neste trabalho mostraremos uma abordagem para especificar como componentes de um sistema interagem entre si. A abordagem dá ênfase à especificação explícita de como os componentes interagem. Nas abordagens tradicionais de especificação, a parte dinâmica da interação é definida implicitamente no comportamento de cada componente, e a parte estática é especificada através das interfaces dos módulos do sistema, com a indicação dos dados e funções que são exportados e importados por um componente.

Como arcabouço para desenvolvimento de nossas idéias utilizamos o método formal ASM (*Abstract State Machines)*[6], o qual é um sistema de transição de estados. Proporemos uma extensão às ASM, a qual chamaremos de IASM (*Interactive Abstract State Machines*). A extensão é baseada na especificação da interação entre

os elementos de computação através de primitivas de troca de mensagens e de configuração da topologia de interconexão estes elementos. Pretendemos mostrar que a nova abordagem é ampla o suficiente para unificar problemas importantes e sem solução plenamente satisfatória em termos formais. Neste trabalho mostraremos que as IASM podem ser aplicada aos problemas de computação móvel e componentes de software.

Na Seção 2 mostramos a linguagem das IASM. É desejável familiaridade com as ASMs. Na Seção 3 revisamos o problema da especificação de computação de sistemas móveis e mostramos que as IASM servem para modelar estes problemas. Na Seção 4 revisamos o problema da especificação de componentes de software e mostramos que IASM também modelam esta classe de problemas naturalmente. Na Seção 5 mostramos as conclusões.

2 Máquinas de Estado Abstratas Interativas

Máquinas de Estado Abstratas Interativas (IASM) são baseadas nas ASM multiagentes [6]. A seguir mostramos uma definição auto-contida das IASM.

- Uma especificação IASM é composta por uma coleção de sub-especificações independentes (definições de unidades).
- Cada sub-especificação é executada por uma coleção de *agentes* de execução (instâncias de unidades).
- Cada sub-especificação tem acesso somente ao seu estado local. Qualquer acesso a informações externas deve ser feito mediante um recebimento explícito da respectiva informação. Assim, o estilo de especificação é baseado em troca de mensagens.
- A troca de mensagens ocorre através de canais de comunicações previamente estabelecidos entre os agentes de execução.
- Os canais de comunicação podem ser estabelecidos estaticamente ou dinamicamente.
- Existe uma separação explícita de dois tipos de computação:
  1. A computação interna é definida por regras que somente modificam o estado local.
  2. A computação da interação é definida por regras que: i) estabelecem a topologia de interconexão entre os agentes de computação, ii) efetivam a troca de mensagens entre os agentes, iii) controlam a execução da computação da interação.

Definição 2.1 Uma *especificação IASM* $\mathcal{SPEC}$ é uma tripla ($\mathcal{CUD}$, $\mathcal{V}$, $\mathcal{M}$), onde:

- $\mathcal{CUD}$ é uma coleção de definições de unidades $\mathcal{UD}$. Cada definição de unidade tem como objetivo encapsular uma parte do estado e uma parte das regras de computação. O ambiente externo de uma definição de unidade é composto pelas definições de unidades com as quais ela interage.
- $\mathcal{V}$ é o vocabulário de toda a especificação. O estilo de especificação que as IASM provê ao projetista é distribuído. Em outras palavras, não é possível para o projetista definir um estado global. Contudo, o modelo semântico

subjacente às IASM considera o estado como uma estrutura global, e portanto com um vocabulário global $\mathcal{V}$.
- $\mathcal{M}$ é a especificação de início que define o estado inicial da máquina. □

Definição 2.2 Uma *definição de unidade* $\mathcal{UD}$ é uma quadrúpla $(\mathcal{N}, \mathcal{IS}, \mathcal{IR}, \mathcal{IC})$ onde:

- $\mathcal{N}$ é o nome da definição de unidade.
- $\mathcal{IS}$ define o vocabulário interno $Fun(\mathcal{UD})$ da definição $\mathcal{UD}$.
- $\mathcal{IR}$ é a declaração da regra de interação a qual guia a interação das respectivas instâncias com seus ambientes. Esta regra define a interface comportamental da definição de unidade $\mathcal{UD}$, estabelecendo como a comunicação entre as instâncias podem ocorrer. Ela define não somente troca de informação, mas também restrições de sincronização impostas entre as instâncias.
- $\mathcal{IC}$ é a declaração da regra interna de computação a qual atualiza o estado local privado da instância. As regras internas de uma definição de unidade tem a mesma sintaxe das regras ASM. □

Um $\mathcal{V}$-**estado** G é um estado de $\mathcal{SPEC}$ se ele satisfaz as restrições impostas em $\mathcal{UD}$, $\mathcal{V}$ e $\mathcal{M}$.

Sejam $\mathcal{IR}_U$ e $\mathcal{IC}_U$ regras de interação e regras de computação interna da definição de unidade U. Seja ++ a concatenação de regras. O **programa** correspondente de `U` é $Prog(U) = \mathcal{IR}_U$ ++ $\mathcal{IC}_U$.

A **execução** de toda especificação é baseada na execução de cada instância de unidade, a qual é definida pelas regras de interação e de computação interna. Uma *execução* de $\mathcal{SPEC}$ pode ser definida exatamente da mesma maneira que as possíveis execuções das ASM multi-agentes, ou seja, é uma coleção parcialmente ordenada de transições de agentes que respeita a condição que, diferentes instâncias não atualizam a mesma porção do estado simultaneamente, e cujas possíveis serializações resultam no mesmo estado.

Para completar a descrição semântica das ASM interativas definiremos as regras de interação e de computação interna. As regras de interação podem ser de entrada e saída, de configuração da topologia de comunicação, de contagem, ou regras ASM convencionais.

2.1 Regras de Entrada e Saída

Seja Out = `c=t -> u` uma *regra de saída*, onde `t` é um termo, `c` é o rótulo do canal de saída, e `u` é um termo representando a instância de unidade destino.

Seja In = `f(tt) <- u.c` uma *regra de entrada*, onde `f` é um nome de função, `tt` é um termo de tipo compatível com a assinatura de `f`, `u` é um termo representando a instância de unidade destino e `c` é o rótulo do canal na instância de unidade destino.

Para um par de regras de entrada e saída ser compatível, os rótulos das regras devem ser iguais.

Proposição 2.1 [Não ordenação das mensagens] Sejam duas instâncias de unidade a e b, previamente conectadas e sem nenhuma sincronização adicional entre elas. Suponha que a envie para b uma mensagem m_i em um estado S_i, e uma mensagem m_j em um estado S_j, ambas através de uma mesma regra r, tal que $i < j$. Então b pode receber, através de uma mesma regra r', a mensagem m_i e m_j em qualquer ordem.

A prova desta proposição pode ser encontrada em [11].

2.2 Regras de Configuração da Topologia

A seguir mostramos a idéia geral do mecanismo de conexão. A conexão sempre é feita entre duas instâncias de unidade que entram em acordo de conexão. Para haver troca de mensagens entre duas instâncias deve haver uma conexão prévia entre elas. Um acordo de conexão entre duas instâncias $a : A$ e $b : B$ ocorre nas seguintes situações:

1	a tenta com b	b tenta com a
2	a tenta com b	b aceita com qualquer $a' : A$
3	a tenta com b	b aceita com qualquer instância
4	a aceita com qualquer $b' : B$	b tenta com a
5	a aceita com qualquer instância	b tenta com a

Uma vez feito o acordo as duas instâncias a e b podem utilizar o canal obtido na conexão para efetuar regras de entrada e saída.

Assim, existem três tipos de regras para estabelecimento de conexão entre duas instâncias de unidade:

1. *Conn1* = `connect c=u:U.s`,
2. *Conn2* = `connect c=U.s`,
3. *Conn3* = `connect c`.

onde `c` é um rótulo de canal definindo o ponto de conexão dentro da instância corrente, `u` é um termo representando a instância a ser conectada, `U` é o nome da definição de unidade da qual `u` foi derivada, `s` é um nome de função que representa um canal em `U`. Quando a conexão é completada `c` deverá ter o valor da instância `u` e `s` deverá ter o valor da instância corrente.

Condição de equidade: Uma unidade u que aceita conexões com instâncias derivadas de uma definição U' mantém a ordem de aceitação das conexões igual à ordem de requisição das conexões. Se em uma mesma transição são requisitadas várias conexões, então a escolha é não-determinista. Como o número de instâncias que podem requisitar conexão em uma mesma transição é limitado então a unidade u têm condições de aceitar todas as conexões em um tempo limitado.

A seguir vamos enunciar uma das propriedades que as regras de conexão obedecem. São satisfeitas propriedades de progresso para os cinco casos possíveis de acordo de conexão. Vamos enunciar apenas para o caso 1.

Proposição 2.2 [Progresso 1] Sejam duas instâncias $a : A$ e $b : B$. Se a tenta conectar com b, e b tenta conectar com a então, em sucessivas transições, ambos atingem um acordo de conexão em um estado com índice finito.

A prova desta proposição pode ser encontrada em [11].

Sejam dois tipos de regras para desfazer a conexão entre duas instâncias de unidade:

1. *DisConn1* = `disconnect c from s`,
2. *DisConn2* = `disconnect c`

onde `c` é um rótulo de canal definindo o ponto de conexão dentro da instância corrente e `s` é um nome de função que representa o canal de conexão da outra instância.

Seja *Alloc* = `new u:U` uma *regra de alocação*, onde `u` é a identificação da nova instância, `U` é a definição de unidade da qual `u` é derivada.

Seja *Dealloc* = `destroy u` uma *regra de desalocação* , onde `u` é a identificação da instância a ser descartada.

2.3 Outras Regras

Seja *Lab* = i : `label` uma *regra rotulada*, onde i é uma regra de interação e `label` é um nome de função de aridade zero, que conta o número de vezes que a interação i foi efetuada.

As regras ASM da parte interativa de uma unidade e as *regras de computação interna* têm a mesma sintaxe e semântica das regras das ASM convencionais.

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 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.

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 seqüencial de regras, ou seja, uma regra da forma $R_1; R_2$, onde primeiro executa-se R_1 e então R_2. O objetivo de não permitir composição seqüencial de regras é possibilitar um modelo simples para provar propriedades de especificações ASM seqüenciais. Dado que uma execução seqüencial é uma seqüência de estados, e que cada transição é definida por *toda* a especificação, é possível provar propriedades por indução no tamanho da seqüência de estados. Este tipo de prova não é possível em uma linguagem de programação imperativa porque cada transição de estado é definida por um comando do programa e não pelo programa inteiro.

3 Modelagem de Computação Móvel

O trabalho pioneiro em modelos semânticos para computação móvel foi o π-cálculo[12], no qual canais de comunicação podem ser transmitidos sobre outros canais, de tal forma que um processo possa adquirir novos canais dinamicamente, dando ao recebedor do canal a habilidade de comunicar sobre o mesmo. Esta abordagem permite abstrair da mobilidade explícita de código e enxergar a mobilidade como sendo um

fato de fazer-desfazer conexões. Tem se tornado uma abordagem comum, a adição de endereços discretos em cálculos de processos e considerar falhas nestes endereços [2]. Estas extensões tem como objetivo tornar explícita a noção de localização de uma computação e a noção de mudança de localização da computação. Em [4], Cardelli argumenta por uma noção explícita de movimentação através de domínios administrativos, definindo o cálculo de ambientes.

Em IASM, nossa abordagem para a mobilidade é baseada nas idéias originais do π-cálculo, ou seja, na construção dinâmica de agentes ativos de computação distribuída e na configuração dinâmica da topologia de comunicação entre estes agentes. É uma abordagem inteiramente distribuída, no sentido, em que toda informação global só pode ser inferida através de interações com o ambiente. Não pré-definimos em IASM a noção de localização, mas que pode ser explicitada manualmente pelo especificador, usando por exemplo a abordagem sugerida em [9].

Suponha uma especificação IASM que tenha as seguintes características:

- Existem três definições de unidade, sejam, `Server1`, `Server2`, `Client`.
- As instâncias derivadas de `Server1` e `Server2` permitem conexão dinâmica com qualquer tipo de unidade para fornecer algum tipo de serviço.
- A definição de unidade `Client` conhece a seção de interação `Server1` e `Server2` e tenta conexão dinâmica com cada uma delas para obter acesso a seus serviços.

Esta especificação IASM pode ser vista como uma abstração de um sistema móvel onde as instâncias `Server1` e `Server2` estão localizadas em sítios de computação diferentes. A tentativa de conexão de uma instância de `Client` com uma instância de `Server1` ou `Server2` pode ser vista como o seu deslocamento para o respectivo sítio para continuar o seu processamento. Desta maneira, obtemos uma abstração simples e geral para a movimentação de agentes. Em [8] mostramos um exemplo de uma especificação mais completa.

4 Modelagem de Componentes de Software

Nesta seção, vamos apresentar dois trabalhos sobre a formalização de componentes de software. O primeiro trabalho, de M. Broy, propõe uma caracterização matemática para arquitetura e componentes de software. O segundo trabalho, de R. Allen e D. Garlan, propõe uma formalização para conexão arquitetural, refinando o conceito de conectores de componentes.

Segundo Broy [3], componentes podem ser modelados formalmente como sistemas interativos que se comunicam através de canais assíncronos. Ele define o conceito de *streams* temporizadas sobre um conjuntos de mensagens para modelar a comunicação através dos canais. Estas *streams* são usadas para denotar as histórias de comunicações através dos canais. Um componente é definido pela sua interface sintática, que é dada pelo conjunto de canais de entrada e saída, e pela sua interface dinâmica que é dada por uma função que mapeia histórias de entrada em um conjunto de histórias de saída. A formalização é simples, baseada em conjuntos e funções. Broy define arquitetura de software como sendo a estruturação de um sistema com o uso de componentes, definindo como eles se conectam e interagem. Formalmente, uma arquitetura é construída através da composição de componen-

tes, mais especificamente, através da composição da função que define a interface semântica.

Segundo Allen e Garlan[1], uma arquitetura de software pode ser definida como uma coleção de componentes computacionais, juntamente com uma coleção de conectores, os quais descrevem as interações entre componentes. Um componente é definido por um conjunto de portas e por sua especificação de função. Uma porta define um ponto lógico de interação do componente com seu ambiente. Um conector é definido por um conjunto de papéis e por uma especificação de *cola*. Os papéis descrevem o comportamento local esperado de cada uma das partes que interagem. Eles especificam as obrigações de cada componente para participar da interação. A especificação da cola define como as atividades dos papéis são coordenadas. É usado um conjunto de CSP de Hoare para definir os protocolos dos papéis, portas e colas e como modelo semântico para especificação completa do sistema.

A seguir fornecemos nossas definições de arquitetura e componentes de software utilizando as IASM.

Em princípio, podemos usar a mesma definição de Broy [3] para sistemas baseados em componentes: sistemas interativos que se comunicam através de canais assíncronos. De uma maneira geral, uma especificação IASM é dada por um conjunto de definições de unidades das quais serão instanciados agentes que executam estas definições. Estes agentes interagem assincronamente entre si, estabelecendo canais de comunicações e trocando informação através destes canais.

Assim, um *componente* pode ser definido diretamente como uma definição de unidade, possuíndo automaticamente as seguintes características:

- Unidade de execução independente. Toda instância de unidade é executada por um agente independente, ativo.
- Comportamento de interface independente. A seção de interação de uma definição de unidade, define mais do que o conjunto de canais de entrada e saída e dos dados de entrada e saída que trafegam sobre estes canais. Ela define também o comportamento da interação, especificando as sincronizações e condições necessárias para cada regra de interação ser disparada.
- Capacidade de composição por terceiros. A possibilidade de configuração dinâmica da topologia de interconexão entre componentes torna a composição de componentes uma tarefa clara pois as regras de interação que estabelecem o protocolo de comunicação estão explicitamente definidas.
- Ausência de dependências de contexto implícita. O fato de não existir estado global e toda interação ser definida explicitamente, todas as dependências entre os componentes são resolvidas de acordo com o contrato de interação definido pelas regras de interação.

Uma *arquitetura de software* é definida pela topologia de interconexão corrente. A possibilidade de especificação da topologia de interconexão de maneira estática ou dinâmica favorece a reestruturação da arquitetura.

Finalmente, ressaltamos que o modelo também é adequado para modelagem de arquiteturas de *software* dinâmicas, nas quais a composição dos componentes muda durante o curso da computação, isto porque permitimos a reconfiguração dinâmica da topologia de interconexão entre os componentes. É interessante notar que esta

reconfiguração dinâmica entre componentes também foi utilizada para modelagem de agentes móveis [8], o que sugere a interseção conceitual [5].

5 Conclusões

O presente trabalho mostrou um modelo semântico formal adequado para duas classes de problemas aparentemente distintas: computação móvel e arquitetura de *software*. Entretanto, estes problemas podem ser modelados adequadamente usando as mesmas abstrações: troca de mensagens e configuração dinâmica da topologia de comunicação entre agentes.

O modelo semântico é baseado em uma abordagem operacional, o que o torna acessível a profissionais de desenvolvimento de software.

O mecanismo de provas a respeito das especificações é definido a partir o conjunto que especifica a execução do sistema. No caso de execuções seqüenciais, o *conjunto* representa uma seqüencia de estados e portanto pode ser utilizada prova por indução matemática. No caso de execuções parcialmente ordenadas, a prova deve ser feita com base nos possíveis conjuntos parcialmente ordenados que são produzidos a partir das regras.

Como trabalhos futuros temos a implementação de um ambiente de execução para as IASM, integração deste ambiente de execução com ferramentas de verificação formal de propriedades, e aplicação da metodologia e ferramentas em estudos de casos de dimensão real.

Referências

[1] R. Allen and D. Garlan. A formal basis for architectural connection. *ACM Transactions on Software Engineering and Methodology*, 6(3):213–249, July 1997.

[2] R. Amadio. An asynchronous model of locality, failure, and process mobility. In *Proceedings of COORDINATION 97*, volume 1282 of *Lecture Notes in Computer Science*. Springer Verlag, 1997.

[3] M. Broy. Towards a mathematical concept of a component and its use. *Software – Concepts and Tools*, 18:137–148, 1997.

[4] L. Cardelli and A. Gordon. Mobile ambients. In M. Nivat, editor, *Foundations of Software Science and Computational Structures*, volume 1378 of *Lecture Notes in Computer Science*, pages 140–155. Springer Verlag, 1998.

[5] P. Ciancarini and C. Mascolo. Software architecture and mobility. In *Third International Software Architecture Workshop (ISAW-3)*, Orlando, Florida, Nov. 1998. ACM Press.

[6] Y. Gurevich. Evolving Algebras 1993: Lipari Guide. In E. Börger, editor, *Specification and Validation Methods*, pages 9–36. Oxford University Press, 1995.

[7] M. Maia and R. Bigonha. The Interactive Abstract State Machine Language. In *I Workshop Brasileiro de Métodos Formais*, Porto Alegre, Outubro 1998.

[8] M. Maia and R. Bigonha. Interaction Based Semantics for Mobile Objects. In *III Simpósio Brasileiro de Linguagens de Programação*, Porto Alegre, Maio 1999.

[9] M. Valente, R. Bigonha, and M. Maia. Aplicação de ASM na Especificação de Sistemas Móveis. *Neste Volume*, Florianópolis, 1999.

[10] M. Maia, V. Iorio, and R. Bigonha. Interacting Abstract State Machines. In *Proceedings of the 28th Annual Conference of the German Society of Computer Science*. Magdeburg University, 1998.

[11] M. Maia. *Especificação Formal da Interação de Componentes de Sistemas Computacionais*. PhD thesis, Universidade Federal de Minas Gerais, Agosto 1999.

[12] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes (Parts I and II). *Information and Computation*, 100:1–77, 1992.

[13] P. Wegner. Interactive software technology. In A. B. Tucker, editor, *Handbook of Computer Science and Engineering*. CRC Press, 1997.