Wanderley LOPES DE SOUZA

GRC/DSC/Universidade Federal da Paraíba
(trabalho realizado junto ao IRO/Université de Montrèal com auxílio fornecido pelo CNPq)

Sumário

O uso de técnicas de descrição formal, para a especificação de software de comunicação, tem sido uma constante fonte de preocupações para os projetistas da área em questão. O objetivo desse artigo é realizar uma análise de alguns trabalhos, que sugerem o emprego de Prolog em diversas fases do desenvolvimento de protocolos, e procurar estabelecer um modelo em Prolog, para a especificação de serviços e protocolos, que facilite as tarefas de validação necessárias ao bom desenrolar do projeto.

1. Introdução

Há pelo menos três passos a serem seguidos no desenvolvimento de software de comunicação:

(a) compreender o problema e formular possíveis metodologias para a sua solucão;

(b) especificar a solução de forma não ambigua, utilizando, por exemplo, uma especificação formal;

(c) implementar a especificação através de uma linguagem de programação.

A conformidade entre essas diferentes etapas pode ser assegurada através de métodos de validação, que variam da verificação formal à execução simbólica e da simulação ao teste. Realizada a última etapa, pode-se dizer que a implementação obtida satisfaz à correspondente especificação, de acordo com as atividades de validação empregadas.

O uso de técnicas baseadas em lógica, para descrever a solução de um problema, apresenta algumas vantagens em relação ao uso de linguagens convencionais:

(a) um mapeamento mais direto entre problema e programa;

(b) o programa pode ser dividido em duas partes: um componente **lógico**, especificando o conhecimento necessário à solução do problema, e um componente de **controle**, definindo como usar esse conhecimento para a resolução do problema;

(c) o programa é de fácil modificação e/ou extensão.

Certas características da linguagem de programação lógica Prolog e o caráter lógico presente nos protocolos de comunicação tem induzido pesquisadores a pensar numa nova aplicação para essa linguagem. Nesse trabalho, após uma apresentação sumária de Prolog e de algumas de suas propriedades, o seu emprego nas fases de especificação e validação de serviços e protocolos é discutido.

2. Linguagem de programação Prolog

Prolog é uma linguagem de programação prática, baseada num modelo próximo ao da programação lógica e na interpretação procedural de Kowalski para cláusulas Horn [Kowa 79].

A principal unidade sintática de Prolog é o **termo**. Um termo pode ser uma constante (inteiro, real ou átomo), uma variável ou um termo composto. O primeiro caráter do identificador de um átomo deve ser uma letra minúscula, enquanto que o de uma variável deve ser uma letra maiúscula. Um **termo composto** consiste de um identificador (iniciado com letra minúscula) seguido de uma lista de **argumentos**. Cada argumento já é em si um termo.

De acordo com a sintaxe apresentada em [ClMe 81], as cláusulas Horn são escritas da seguinte forma:

conclusao :- condicao1, condicao2, ..., condicaoN.

onde conclusão e condições são termos compostos do tipo:

termo (arg1, arg2, ..., argM)

A **cláusula** acima define uma **relação** entre conclusão e condições, cuja interpretação é a seguinte: **conclusao**, que é a **cabeça** da cláusula, será verdadeira se (":-") **condicao1** e (",") **condicao2** e ..... e **condicaoN**, que constituem o **corpo** da cláusula, forem verdadeiras. O símbolo "." indica o fim da cláusula.

Uma cláusula sem corpo corresponde a uma **asserção** e será sempre verdadeira. Uma cláusula sem cabeça é interpretada como uma **meta** (ou questão) e nessa sintaxe deve ser iniciada com o símbolo "?-".

Um programa em Prolog é constituido de um conjunto de cláusulas Horn. A execução desse programa corresponde a uma prova, ou sucessão de deduções, de uma dada **meta global** (?- questao1, questao2,....., questaoK.). Tal prova é realizada através de uma pesquisa sequencial no programa, associada a "**backtracking**", visando a unificação das submetas, presentes na meta global, às cláusulas desse programa. O resultado da execução pode ser **sucesso ou falha**. No caso de sucesso, os valores dos parâmetros presentes na meta global constituem a saída da execução.

Em Prolog, o grau de não determinismo, presente no modelo de programação lógica de [Kowa 79] é reduzido. Por questões de simplicidade e eficiência, submetas e cláusulas são escolhidas na pesquisa sequencial, da esquerda para a direita e de cima para baixo respectivamente.

A lista é uma das principais estrutura de dados em Prolog. Uma lista é uma sequência de elementos ordenados, que pode ter qualquer comprimento. Os elementos de uma lista podem ser termos ou listas. Uma lista é frequentemente expressa utilizando-se o operador "|", os delimitadores "[" e "]", da seguinte forma:

[cabeca|cauda]

onde **cabeca** é sempre um elemento e **cauda** uma lista. Uma lista vazia é representada por [].

Estruturas de dados, predicados e estruturas de controle são comumente definidos em Prolog de forma recursiva. Por exemplo, o predicado **acrescente**, que adiciona ao final de uma lista [X|L1] uma lista L2, gerando uma nova lista [X|L3], pode ser definido da seguinte forma:

```
acrescente ([], L, L).
acrescente ([X|L1], L2, [X|L3]) :- acrescente (L1, L2, L3).
```

As cláusulas de um programa em Prolog podem ser interpretadas de forma **declarativa** ou **procedural**. No primeiro caso, tais cláusulas definem objetos e relações entre objetos, podendo o programa ser interpretado como uma **linguagem de especificação** [Davis 85]. No exemplo acima, a primeira cláusula informa que acrescentar a uma lista vazia uma lista resulta na própria lista. A segunda, entre outras informações, especifica que será acrescentada a uma lista não vazia uma segunda lista, gerando uma terceira com a cabeça da primeira, se à cauda da primeira for acrescentada a segunda, gerando a cauda da terceira.

Na semântica procedural define-se como o programa é executado. As cláusulas podem ser vistas como uma base de dados e são executadas como funções, quando uma submeta a ser satisfeita é escolhida. No exemplo acima, o conjunto de cláusulas pode ser usado pelo menos de dois modos:

(a) como **reconhecedor** de relações. Por exemplo, se a seguinte meta é escolhida

```
?- acrescente ([a], [b,c], [a,b,c]).
```

o resultado da execução será **sucesso**;

(b) como **gerador** de respostas que satisfazem a uma determinada relação. Por exemplo, se a seguinte meta é escolhida

```
?- acrescente (Y, Z, [a]).
```

a primeira resposta será Y = [] e Z = [a]. Caso uma nova alternativa seja solicitada, a segunda resposta será Y = [a] e Z = [].

Estruturas de controle podem também ser especificadas em Prolog, utilizando-se recursividade. Por exemplo, um **laço** pode ser definido da seguinte forma:

```
execute (Inicio, Fim) :- predicado (Inicio, Seguinte),
                         execute (Seguinte, Fim).

execute (Inicio, Fim) :- predicado (Inicio, Fim).
```

a primeira cláusula **execute** gera o laço, enquanto a segunda permite parar a execução desse laço.

3. Especificação de serviços e protocolos em Prolog

O desenvolvimento de serviços e protocolos de comunicação é normalmente realizado através de uma sequência de fases, iniciando-se com a formulação de intenções (**especificação informal**) e convergindo para uma **implementação**. Uma das fases intermediárias é a **especificação formal**, que tem por objetivo a descrição clara e concisa de serviços e protocolos. As **especificações de serviços** descrevem os serviços fornecidos pelos níveis hierárquicos de protocolos e as **especificações de protocolos** descrevem o comportamento das entidades a serem implementadas.

Máquinas de estado finito (MEF) são frequentemente utilizadas como forma de representação do comportamento de serviços e protocolos. Os elementos constituintes de uma MEF são **estados** (estados principais) e **interações**. Uma MEF ao receber uma interação de **entrada** (proveniente de seu ambiente) ou **espontânea** (interna) executa uma **transição**. Ao executar uma transição a MEF mudará (ou não) de estado, assim como produzirá (ou não) uma interação de **saída** para o seu ambiente.

A fim de permitir a introdução dos **parâmetros** presentes nas interações (**primitivas de serviço ou unidades de dados de protocolo**) e a fim de introduzir **variáveis de estado adicionais**, as MEF são estendidas (**MEFE**). Isso permite a descrição completa das transições, que se contituem no corpo das especificações formais de serviços e protocolos, representados através de MEFE. Um exemplo de linguagem de especifição baseada em MEFE é apresentado em [Lopes 85].

O uso de programação lógica e Prolog para a especificação de serviços e protocolos é proposto em [LoSiUr 84] e [Sidhu 83] respectivamente. Ambas as especificações, embora realizadas com objetivos diferentes e a partir de modelos diferentes, contêm duas partes distintas: uma **lógica** e outra de **controle**. No componente lógico dessas especificações a forma utilizada para a descrição das transições é semelhante a:

```
trans (Estado_atual, Estado_seguinte, Entrada, Saida)
```

onde as variáveis **Estado_atual** e **Estado_seguinte** representam os estados principais antes e depois da transição respectivamente; **Entrada** e **Saida** representam as interações (sem os seus parâmetros).

A fim de permitir a inclusão das variáveis de estado adicionais e dos parâmetros das interações, nós propusemos o seguinte modelo em [BoDsLoSaUr 85]:

```
trans (Estado_atual, Estado_seguinte, Entrada, Saida) :- condicao, acao.
```

onde os argumentos da **cláusula trans** são listas. Os dois primeiros são

constituidos de um estado principal seguido de variáveis de estado adicionais e os dois últimos são constituidos de uma interação seguida de seus parâmetros. Por exemplo:

```
Estado_atual = [fechado, Em_uso, Opcoes, .....]

Entrada = [t_CONreq, Para_endereco, Opcoes_propostas, .....]
```

O corpo da cláusula **trans** é constituido de dois conjuntos de predicados, isto é, **condicao** e **acao**. O primeiro habilita a transição e o segundo atualiza algumas variáveis de estado (as demais são atualizadas diretamente em Estado_seguinte).

Para a especificação das transições de um módulo composto de um conjunto de submódulos, por exemplo, duas entidades de protocolo e um módulo de serviço (Fig.1), a estrutura das variáveis, que representam os estados atual e seguinte desse sistema, pode ser do tipo:

```
[Estado_entidade1, Estado_entidade2, Estado_servico]
```

A estrutura das variáveis, que representam as interações de entrada e de saída, pode ser do tipo:

```
[Interacao_servico1, Interacao_servico2]
```

sendo que as interações ocorrem nos **pontos de acesso ao serviço_N**, isto é, PASN1 e PASN2 (Fig.1).

Uma vez que cada submódulo do sistema suporta duas entradas e duas saídas, o modelo de transição apresentado (**cláusula trans**) deve ser ligeiramente modificado para:

```
t (Estado_atual, Estado_seguinte, [Entrada1, Entrada2], [Saida1, Saida2])
:-
condicao, acao.
```

onde a ausência de uma entrada ou saída é representada por [].

As **transições globais** do módulo apresentado na Fig.1, que fornece o serviço_N e onde as interações são do tipo "rendez-vous" (isto é, entrada e saída simultâneas nas interfaces), podem ser descritas através do modelo:

```
tSN ([E1a, E2a, ESNIa], [E1s, E2s, ESNIs], [ESN1, ESN2], [SSN1, SSN2])
:-
t (E1a, E1s, [ESN1, SSNI1], [SSN1, ESNI1]),
t (E2a, E2s, [ESN2, SSNI2], [SSN2, ESNI2]),
tSNI (ESNIa, ESNIs, [ESNI1, ESNI2], [SSNI1, SSNI2]).
```

Se um modelo com **filas de interação** for utilizado (por exemplo, filas entre as entidades e o submódulo de serviço da Fig.1), às variáveis de estado, presentes nos argumentos da cláusula **tSN**, devem ser acrescentadas variáveis representando os estados das filas. Ao corpo da cláusula **tSN** devem ser introduzidos predicados **acrescente** e **retire** (semelhantes ao definido na seção 2), que serão responsáveis pelas operações a serem realizadas nessas filas e pelas atualizações de seus estados.

No modelo de transição global proposto, uma transição do módulo consiste de transições simultâneas de seus submódulos. Caso filas de interação sejam utilizadas, é possível definir-se um outro modelo, onde uma transição de um dos submódulos acarreta numa transição global do sistema. Nesse caso, a cláusula tSN deverá ser constituida de um conjunto de cláusulas alternativas, uma para cada submódulo [UrPr 84], ou uma para cada tipo de transição dos submódulos [Sidhu 83].

4. Utilização de Prolog para a validação de serviços e protocolos

O objetivo da validação é verificar se representações de um mesmo sistema, apresentadas em diferentes níveis de abstração, são conformes. As atividades de validação são normalmente divididas em duas categorias: verificação e teste.

Verificação procura demonstrar logicamente propriedades dos serviços e protocolos, tais como:

(a) ausência de impasses ("deadlock freeness"): o sistema nunca atingirá um estado, a partir do qual não será possível uma nova transição;

(b) ausência de ciclos improdutivos ("livelock or tempo-blocking freeness"): não ocorrerão ciclos improdutivos entre qualquer subconjunto de estados do sistema;

(c) finalização ("termination"): a partir de um estado, existirá sempre um "caminho" através dos estados, permitindo ao sistema atingir um estado final previsto na especificação.

Teste procura demonstrar propriedades semelhantes, mas através da execução controlada do sistema e comparando o comportamento observado em relação ao especificado.

Especificações de serviços e protocolos baseadas em MEFE podem ser verificadas, em relação às propriedades acima descritas, através da geração de caminhos de interação pertencentes ao espaço de estados do sistema representado. Em [Sidh 83], o componente de controle da especificação do sistema é constituido de um conjunto de cláusulas, que permitem verificar se existe um caminho de interação entre um estado inicial e um estado meta desejado. Em [LoSiUr 84], o componente de controle explora as propriedades de reconhecimento e geração da linguagem Prolog (descritas na seção 2), para reconhecer se uma sequência de interações é válida, ou para gerar todas as possíveis sequências válidas de interação a partir de um determinado estado do sistema.

Sequências de teste para serviços e protocolos são apresentadas em [UrPr 83]. Tais sequências são especificadas numa gramática livre de contexto atribuida (para permitir a inclusão dos parâmetros das interações e alguns parâmetros de controle). A gramática obtida é então implementada em Prolog e a implementação é executada, de forma controlada, a fim de gerar as sequências de teste. Entretanto, essa especificacção, por não conter a noção de estado global do sistema, pode ser usada somente como geradora e não como reconhecedora de sequências válidas de interação.

Os artigos acima citados procuram demonstrar as vantagens de

dispor-se de especificações executáveis de sistemas e em particular as vantagens propiciadas pela programação lógica e Prolog, ao permitirem a divisão de uma especificação numa parte lógica e outra de controle. Por exemplo, a execução de uma sequência de transições pode ser descrita através de uma estrutura de controle, semelhante ao laço apresentado na seção 2, da seguinte forma:

```
execute (Estado_inicial, Estado_final,
        [Entrada|Entrada_cauda], [Saida|Saida_cauda])
:-
trans (Estado_inicial, Estado_seguinte, Entrada, Saida),
execute (Estado_seguinte, Estado_final, Entrada_cauda, Saida_cauda).

execute (Estado_inicial, Estado_final, [Entrada], [Saida])
:-
trans (Estado_inicial, Estado_final, Entrada, Saida).
```

A primeira cláusula indica que um **caminho de interação**, partindo de um **Estado_inicial** e terminando num **Estado_final**, é possível, se:

(a) existe um predicado **trans**, cujas **Entrada** e **Saida** correspondem respectivamente ao primeiro elemento da sequência de entrada e ao primeiro elemento da sequência de saída, que partindo de um **Estado_inicial** leva a um **Estado_seguinte** e

(b) existe um possível **caminho de interação remanescente**, cujas sequências de entrada e saída correspondem respectivamente às caudas das sequências anteriores, que partindo do **Estado_seguinte** leva ao **Estado_final**.

Esse componente de controle pode ser usado para:

(a) **simular execução**: a partir de um estado inicial e de uma dada sequência de interações de entrada, determina-se os possíveis estados finais com suas respectivas sequências de interações de saída;

(b) **verificar um traço**: a partir de um estado inicial e de um traço observado (sequência de entradas e saídas) de uma implementação, determina-se se o traço é válido em relação à especificação e determina-se os possíveis estados finais da implementação, após a execução das transições correspondentes a esse traço;

(c) **encontrar caminhos de interação**: a partir dos estados inicial e final, determina-se as possíveis sequências de entrada e saída correspondentes às possíveis sequências de transições, que levam do estado inicial ao estado final.

Variações dessa estrutura de controle, que permitem, por exemplo, a definição de uma **história de execução** (sequência de elementos da forma [Estado, Entrada, Saida]), são apresentadas em [BoDsLoSaUr 85].

Um sistema, que pode utilizar Prolog como ferramenta para testes de serviços e protocolos de comunicação, é descrito em [LoDsBo 85]. Tal sistema é constituido basicamente de três módulos: o primeiro especificando uma referência para os testes, o segundo é um gerador de sequências de teste e o último um reconhecedor de sequências de interações provenientes da implementação sob teste. Em Prolog, esse sistema poderia corresponder à definição de um componente lógico, obtido a partir da

especificação do serviço ou protocolo, e de dois (ou um somente) componentes de controle responsáveis pela geração e reconhecimento das sequências de interações.

5. Conclusão

Nesse trabalho, uma análise de algumas possibilidades de utilização de Prolog na área de Protocolos de Comunicação é realizada. Um modelo em Prolog para as transições de sistemas, cujas especificações são baseadas em Máquina de Estado Finito Estendida, é proposto. Em seguida, um exemplo de uma estrutura de controle, que permite explorar (para fins de validação) a descrição das transições de um módulo, é apresentado.

As grandes vantagens de especificações escritas em Prolog parecem residir no fato de serem executáveis e de serem reversíveis, isto é, é possível definir-se um componente lógico e outro de controle, que permitam a geração e reconhecimento de sequências de interações, facilitando assim as tarefas de verificação e teste de serviços e protocolos de comunicação.

Como proposta para trabalhos futuros, sugerimos a especificação de um exemplo completo, utilizando os modelos em Prolog apresentados nesse artigo, e submeter esse exemplo a uma série de métodos de validação. Isto poderá permitir uma melhor visualização das condições de aceitabilidade de Prolog, como linguagem de especificação de serviços e protocolos, sobretudo no que tange às questões de desempenho dessa linguagem.

6. Referências

[BoDsLoSaUr 85] : G.V. Bochmann, R. Dssouli, W. Lopes de Souza, B. Sarikaya, H. Ural, "Use of Prolog for Building Protocol Design Tools", publicação interna No 524 (submetida ao Fifth Int. Workshop on Protocol Specification Verification and Testing), Univ. de Montréal, março 1985.

[ClMe 81] : W.F. Clocksin, C.S. Mellish, "Programming in Prolog", Springer- Verlag, 1981.

[Davis 85] : R. Davis, "Executable Specifications as Basis of Program Life Cycle", anais do 18th Hawaii Int. Conf. on System Sciences, vol II, pp. 722-733, jan 1985.

[Kowa 79] : R. Kowalski, "Logic for Problem Solving", Elsevier North-Holland, 1979.

[LoDsBo 85] : W. Lopes de Souza, R. Dssouli, G.V. Bochmann, "Ambiente de Teste para Protocolos de Comunicação", anais do 3 Simpósio Brasileiro de Redes de Computadores", abril 1985.

[Lopes 85] : W. Lopes de Souza, "Utilização dos Conceitos de Módulo, Porta e Canal em Especificações de Serviços, Protocolos e Interfaces de Comunicação", anais do 3 Simpósio Brasileiro de Redes de Computadores, abril 1985.

[LoSiUr 84] : L. Logrippo, D. Simon, H. Ural, "Executable Description

of the Transport Service in Prolog", anais do Fourth Int. Workshop on Protocol Specification Testing and Verification, North-Holland, 1984.

[Sidhu 83] : D.P. Sidhu, "Protocol Verification via Executable Logic Specifications", Protocol Specification, Testing and Verification, III, North-Holland, pp. 237-248, 1983.

[UrPr 83] : H. Ural, R.L. Probert, "User-guided Test Sequence Generation", Protocol Specification, Testing and Verification, III, North-Holland, pp. 421-436, 1983.

[UrPr 84] : H. Ural, R.L. Probert, "Testing Specifications and Designs of Communication Protocols and Services", publicacao interna TR-84-18, University of Ottawa , 1984.

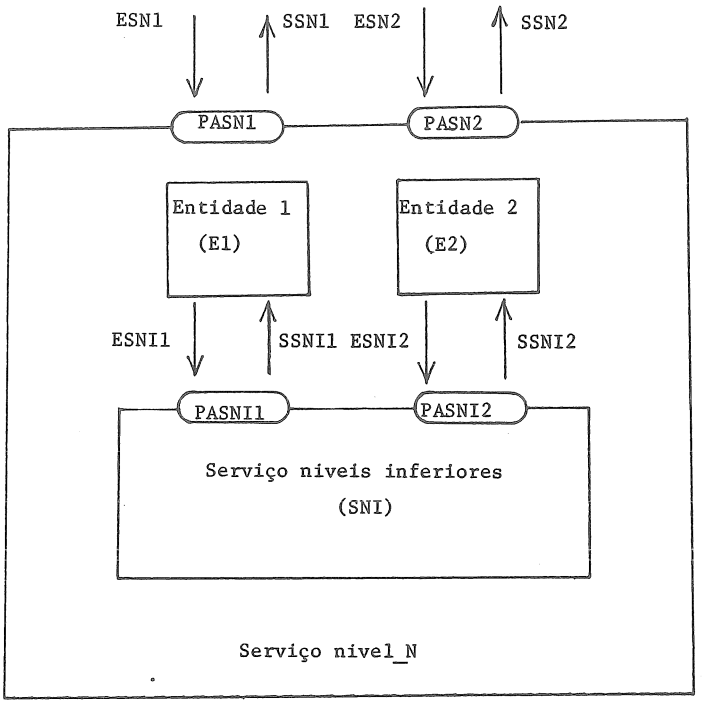

Fig. 1. Sistema composto fornecendo o serviço do nivel_N