Uma Ferramenta para Análise e Verificação de Protocolos Especificados em SDL

Rodrigo Stumpf Trindade
stumpf@inf.ufrgs.br
Liane Margarida Rockenbach Tarouco
liane@penta.cesup.ufrgs.br

Pós-Graduação em Ciência da Computação
Instituto de Informática
Universidade Federal do Rio Grande do Sul
CP 15064, CEP 91501-970 - Porto Alegre - Brasil

Sumário

Este trabalho descreve uma ferramenta para análise e verificação de protocolos baseada na transformação da linguagem SDL em PROLOG. Utilizando este ambiente para a especificação de sistemas de comunicação, é possível verificar propriedades desejáveis, tais como ausência de deadlocks, assim como analisar as possíveis seqüências de transições na máquina de estados finitos de um protocolo. Como exemplo, é utilizado o protocolo TCP, por sua ampla e difundida utilização.

Abstract

This work describes a tool for analisys and verification of protocols based on the transformation of the SDL language into PROLOG. Using this environment for the specification of communications systems, it is possible to verify desirable properties, such as deadlock absence, as well as to analyse the possible transitions sequences in a protocol finite state machine. As an example, it is used the TCP protocol, for his wide and diffused utilization.

1 Introdução

A especificação formal de protocolos requer o uso de uma linguagem com referências a dados (objetos), e para que possa ser utililizada por algum tipo de ferramenta de verficação automática, deve atender ao pré-requisito de possuir uma semântica formalmente definida. O objetivo do desenvolvimento de um ferramenta é prover uma linguagem onde seja possível expressar operações matemáticas (sobre estruturas, listas, por exemplo) de uma maneira não-procedural possibilitando a verificação e análise de protocolos.

O embasamento primário decorre de estudos [TRI 91] e [TRI 92] envolvendo SDL [Z.100] e no desenvolvimento de projetos e protocolos de comunicação de dados. Podem ser verificadas propriedades estáticas e dinâmicas dos protocolos. Geralmente inicia-se a verificação por uma das entidades (um dos lados) isoladamente. Somente então parte-se para a análise do conjunto. A necessidade de verificação formal pode ser considerada secundária pois mesmo para um protocolo de estabelecimento de conexão simples, a verificação está longe de ser totalmente automática. Um entendimento do protocolo é essencial na determinação da estrutura do sistema de provas e na definição das propriedades das entidades que o compoem.

Entretanto, se propriedades básicas são expressas em uma técnica de especificação, então o processo de verificação torna-se consideravelmente simplificado. Neste caso, a possibilidade de verificação semi-automática de sistemas com autilização da intervenção humana é promissora.

A idéia básica da ferramenta proposta neste trabalho é prover um ambiente para a declaração e execução de especificações, de a forma a propiciar uma simulação do comportamento de softwares de protocolos.

O objetivo do uso da linguagem PROLOG é duplo: o mecanismo de inferência permite que se possa gerar as seqüências de transições ou verificar se as mesmas são válidas para uma dada especificação. PROLOG é uma linguagem declarativa e executável por natureza, e pode ser utilizada para expressar vários modelos de comportamento, como máquinas de estados finitos. O modelo de transição de estados de máquinas de estados finitos (*finite state machines* - FSM) presente em SDL é utilizado.

Neste contexto, a contribuição deste trabalho está consolidada no provimento de um ambiente útil e produtivo tanto para a análise como verificação de protocolos.

2 Análise e Verificação de Protocolos

Normalmente são utilizadas técnicas de especificação formal para auxiliar a análise e verificação de protocolos. Alguns dos objetivos destas técnicas são [ISO 81]:

- especificação sem ambigüidades, claras e concisas;
- uma base para a análise de critérios tais como eficiência e correção;
- suporte direto para a implementação.

Uma técnica de descrição formal deve prover o suporte para construção de especificações com as seguintes características:

- modular, isto é, deve ser possível compor especificações a partir de módulos menos complexos;

- abstrata, isto é, uma técnica de descrição formal deve oferecer a possibilidade de abstrair da especificação detalhes irrelevantes.

Os princípios de modularidade e abstração são úteis para gerenciar o tamanho de sistemas complexos, como protocolos e sistemas de comunicação.

Há vários usos possíveis para especificações como, por exemplo, servir de base para uma análise do protocolo possibilitando responder algumas perguntas típicas tais como:

- Uma determinada seqüência de primitivas do protocolo é válida ou não?
- Quais são alguns exemplos de seqüências de primitivas válidas permitidas pelo serviço especificado? (oposto da questão anterior).
- Dada uma determinada seqüência de primitivas gerada pelo usuário, quais são os possíveis comportamentos do provedor do serviço?
- Quais são alguns exemplos de comportamento do usuário que levam a um determinado comportamento do provedor do serviço? (oposto da questão anterior).

A verificação pode ser considerada como uma prova formal de que uma especificação satisfaz determinadas propriedades como, por exemplo, ausência de deadlocks ou livelocks, utilizando métodos matematicamente justificáveis.

2.1 Propriedades de Protocolos

Várias classificações de propriedades de protocolos são encontradas na literatura [SAJ 84]. Basicamente, podem ser divididas nas seguintes classificações:

- gerais, isto é, comum a todos os protocolos;
- específicas, como por exemplo a execução correta de acordo com o propósito de um protocolo;
- correção parcial, isto é, a consistência da ação executada pelo protocolo em relação à especificação do serviço;
- finalização ou progresso, ou seja, a execução de um serviço em um interval de tempo finito;
- propriedades sintáticas, tratam da estrutura lógica da troca de mensagens (ausência de recepção de sinais não especificados, sinais nunca recebidos, ambigüidades de estado, deadlock estático);
- propriedades semânticas fracas, tratam do progresso efetivo de um protocolo (ausência de deadlocks dinâmicos);

- propriedades semânticas fortes, tratam do comportamento esperado de um protocolo (recuperação de erros).

Uma lista com definições informais de propriedades de protocolos pode ser vista a seguir. Tais propriedades são aplicáveis tanto a protocolos que iniciam em um estado e terminam em outro especificado como final (*protocolos terminantes*), assim como *protocolos cíclicos*, que nunca terminam, e cujo estado final é alcançado repetidamente.

- *segurança* significa que um protocolo nunca entra em um estado inaceitável, onde alguma ação imprevisível possa ocorrer. Esta propriedade engloba a correção parcial, ausência de deadlocks e exclusão mútua.

- *regularidade* significa que um protocolo eventualmente entra em um estado indesejável, mas há a possibilidade para que algo dentro do previsto ocorra. Inclui a finalização ou progresso, e a correta execução do objetivo do protocolo.

- *exclusão mútua* determina que certas ações em ambos os lados de um protocolo não podem ser executadas simultâneamente, ou seja, as máquinas de estado de cada lado do protocolo não podem requisitar o mesmo recurso ao mesmo tempo.

- *correção parcial* significa que um protocolo garante o provimento do serviço requerido caso alcance o seu estado final (no caso de protocolos terminantes), ou inicial (no caso de protocolos cíclicos).

- *limitado* significa que durante a operação de um protocolo o número de mensagens em cada canal entre entidades não pode exceder a capacidade do canal.

- *ausência de sobre-especificação* entende-se como sendo a falta de partes redundantes em especificação de protocolo, ou seja, existência de recepções de sinais nunca transcorridas.

- *completo* denota um protocolo cuja especificação contém a descrição de todas as possíveis entradas, e portanto é livre de recepções inesperadas de mensagens.

- *livre de deadlock estático* ou ausência de deadlock significa que um protocolo nunca entra em um estado no qual todas as filas de entrada de sinais estão vazias e nenhuma transmissão é possível; não havendo possibilidade de progresso.

- *livre de livelock* é um protocolo que nunca entra em um estado no qual ciclos improdutivos são possíveis e a transferência das mesmas mensagens se repete finita ou infinitamente.

- *recuperação de falhas*, auto-sincronização ou estabilidade significa que após uma situação anormal, um protocolo retornará ao um estado normal após um intervalo finito de tempo. Esta propriedade está relacionada à finalização.

- *finalização ou progresso* é a execução do serviço requerido em um intervalo finito de tempo. A finalização é aplicável a protocolos terminantes e significa que um protocolo sempre alcança seu estado final. O progresso aplica-se a protocolos cíclicos, e significa que o protocolo sempre alcança seu estado inicial.

- *imparcial* diz-se de um protocolo no qual cada entidade do protocolo tem a possiblidade de realizar progresso, independentemente do que a outra entidade esteja fazendo.

- *correção total* significa que um protocolo é parcialmente correto e adicionalmente alcança seu estado final (no caso de protocolos terminantes) ou estado inicial (no caso de protocolos cíclicos), para todas possíveis seqüências de entradas (mensagens ou comandos do usuário).

3 Técnicas de Análise e Verificação

Os principais métodos para análise e verificação de projetos de protocolos citados em [BOC 84] são:

- Análise de alcançabilidade, utilizada principalmente no caso de especificações na forma de máquinas de estados finitos;

- Derivação de invariantes, utilizada para determinar os estados alcançáveis baseados na estrutura de especificações dadas na forma de Redes de Petri;

- Prova de programas, utilizado quando um protocolo é especificado em uma linguagem de programação;

- Execução simbólica, similar a prova de programas, também trata de protocolos sob a forma de programas, e analisa todas as possíveis transições a serem executadas.

Neste projeto são utilizados os métodos de execução simbólica e análise de alcançabilidade, uma vez que o modelo da linguagem SDL é baseado no conceito de máquinas de estados finitos.

Na figura 1, pode ser visto um resumo abrangente das principais técnicas utilizadas para garantir determinadas propriedades em protocolos.

Uma classificação mais simples considerando apenas os principais grupos de técnicas de verificação de protocolo divide-se em seis grupos: análise de alcançabilidade, técnica híbrida, prova de assertivas, lógica temporal, simulação e execução da especificação.

A potencialidade de verificação de cada técnica pode ser estabelecida, a grosso modo, pelo número de propriedades que podem ser provadas por tal técnica. A

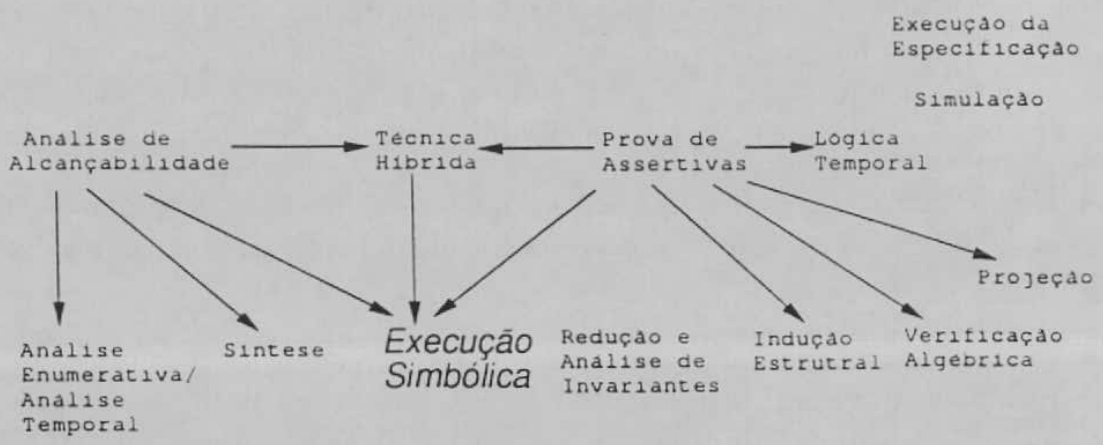

Figura 1: Técnicas de Verificação de Protocolos

importância de cada propriedade pode ser maior ou menor conforme o tipo e modelo do protocolo.

Em [SAJ 84], uma classificação ampla das técnicas de verificação é dada, incluindo as citadas:

- *Análise de Alcançabilidade.*
  Consiste na exploração de todas as possíveis interações entre entidades de um protocolo, através da construção de uma árvore de alcançabilidade a partir estado inicial global. Cada novo estado global (ou seja, o estado de ambas as entidades e os canais) é gerado como resposta a um possível evento.

- *Técnica Híbrida.*
  Utiliza assertivas de programas para as provas de correção, sendo que o modelo de estados que provê a informação necessária para verificação de propriedades gerais. Há pelo menos duas maneiras possíveis de utilização desta técnica. É possível estabelecer assertivas aplicadas a diagramas de transição de estados e variáveis de programa, ou utilizar invariantes indutivas para derivar as assertivas que combinadas com o grafo do fluxo computacional de execução podem provar que transições indesejáveis são impossíveis. Esta técnica usa o modelo de transições ou grafos para especificar comportamento.

- *Prova de Assertivas*
  É um método aplicado à verificação de programas equivalentes a especificações. A verificação de programas implica em assertivas introduzidas em certos pontos do conjunto de programas que implementam a especificação formal de um protocolo de comunicação. Tais assertivas são verificadas a partir de outras que são derivadas da especificação do serviço, expressando as propriedades desejadas do protocolo. São utilizadas linguagens de alto nível tais como Pascal, Algol, CSP.

- *Lógica Temporal*
  É uma extensão da lógica de predicados incluindo operadores que tratam de seqüências de eventos totalmente ordenados no tempo. Um protocolo é especificado como um programa com módulos concorrentes e sua verificação pode ocorrer de três maneiras distintas. A primeira abordagem é similar à verificação de programas, sendo o sistema que constitui o protocolo um conjunto de módulos comunicantes. Outra abordagem utiliza lógica temporal baseada em eventos. Seqüências de interações representam a comunicação entre módulos e ambiente. A terceira abordagem é baseada na idéia de transformação para uma linguagem de alto nível que possa ser analisada de forma a verificar a sua conformidade de acordo com a especificação.

- *Simulação*
  Consiste na representação do comportamento de um protocolo de comunicação pela execução de um programa. Este programa é de fato o modelo executável da máquina do protocolo. Este método é bastante utilizado nas estimativas de performance e medidas de desempenho a partir das estatísticas providas pela execução do programa.

- *Execução da Especificação*
  Consiste na execução de um programa escrito diretamente em uma linguagem de especificação executável. Deve ser provido um interpretador ou tradutor para a linguagem de especificação formal, permitindo a verificação de propriedades desejáveis ao protocolo. Esta execução ainda fornece um teste na especificação antes da implementação, sendo também conhecida como prototipagem rápida, uma técnica bastante útil na verificação de protocolos de comunicação.

Pelo fato de SDL utilizar o conceito de máquinas de estados finitos, naturalmente a técnica de análise de alcançabilidade se apresenta como uma forte escolha. O uso de PROLOG, através da transformação de uma especificação SDL, é suportado pela teoria obtida a partir das técnicas de execução da especificação e execução simbólica.

4 Análise de Especificações

Um dos principais objetivos nas metodologias de engenharia de software é a detecção de erros o mais cedo possível, considerando o tempo dispendido durante as fases do ciclo de vida, de forma a tornar mais barata a sua correção.

É possível através da inspeção sistemática de especificações SDL encontrar erros tais como a falta de um INPUT para uma ação de OUTPUT de um sinal, estados que não podem ser alcançados, sinais salvos por um comando SAVE mas nunca efetivamente recebidos. As principais vantagens da análise de especificações são a relativa facilidade e a possibilidade de garantir a ausência de determinadas classes de erros.

Através da execução de especificações, é possível realizar análises completas ou incompletas. Uma análise completa significa que cada uma das possíveis transições no espaço de estados é executada no mínimo uma vez com cada permutação da população de sinais de entrada disponíveis acrescidos às variáveis. Quanto maior a complexidade do sistema especificado, maior é a chance de overflow do espaço de estados, mas pode ser reduzido através do desenvolvimento de algoritmos para busca e *backtracking*. O problema da população de variáveis continua sendo bastante complexo e de difícil solução.

A execução de uma especificação envolve dois passos:

1. Transformação da especificação em um programa executável.
2. Execução deste programa sobre um ambiente de suporte adequado.

Uma especificação pode ser executada de duas formas:

- como uma análise randômica do espaço de estados, isto é, a interpretação de uma especificação baseado na transição entre estados e comportamento aleatório;
- como uma análise controlada pelo usuário, que pode ser comparado com as metodologias tradicionais, na qual o usuário fornece as sequências de teste.

A maior vantagem do primeiro método mencionado é a capacidade de análise sem que seja necessário um cenário explícito de testes. Além disto, a análise pode ser iniciada tão logo comece a fase de projeto. No caso de uma análise do comportamento de um sistema sob condições pré-determinadas, tais como uma população de variáveis elevada ou uma combinação pouco provável de sinais, então o controle do usuário é demandado.

5 A Arquitetura do Sistema

Nesta seção será descrita a semântica operacional de processos PSDL ($PROLOG \leftarrow SDL$). Para tanto será feito um mapeamento de SDL para um sistema de transições [ORA 88].

Um sistema de transições está em um estado que pode ser alterado por eventos. A transmissão e a recepção de mensagens pode ocorrer simultâneamente com transições entre estados. Um estado consiste tipicamente de valores de variáveis internas, conteúdo de buffers de mensagens, etc. A representação de uma mensagem é feita através de eventos de comunicação, definidos como ações atômicas. Tais eventos podem ser sinais de entrada ou de saída.

Cada processo PSDL pode ser representado como um sistema de transições, definido como uma tupla $< I, O, S, S_0, T >$, onde:

- I é o conjunto de sinais de entrada (*i_signal*);

- O é o conjunto de sinais de saída (*o_signal*);
- S é o conjunto de estados (*state*);
- S_0 é o estado inicial, $S_0 \in S$;
- T é um subconjunto de $SXSX(I \cup O)$, definido como *transições PSDL*, denotadas por $S_1 \rightarrow S_2$, onde $S_1, S_2 \in S$ (*trans(P,S1,S2,I,O)*).

O uso de PROLOG é bastante sugerido em diversas áreas envolvendo protocolos de comunicação de dados [KOY 87], [CAR 89], [UNR 87], basicamente pelas seguintes características:

- executável, graças ao mecanismo de inferência PROLOG, é possível executar uma especificação diretamente, sem adições.
- não-procedural, é possível especificar um sistema através de fatos PROLOG.
- pode ser usada como gerador ou reconhecedor de comportamento de um protocolo.

Para implementar a verificação e análise são utilizadas técnicas de execução da especificação e execução simbólica, e também a análise de alcançabilidade. Baseado nestes conceitos, decorre a arquitetura da ferramenta proposta (figura 2).

A transformação entre as linguagens é abordada em [UNR 87]. Uma especificação PSDL é obtida pela transformação manual entre as linguagens (exemplo figura 3) podendo ser implementada na própria linguagem PROLOG, através de um *parser*.

Este trabalho está concentrado na tentativa de realizar análise com a intervenção humana e verificações automáticas em especificações PSDL.

Assim, um conjunto de fatos PROLOG é utilizado para implementar a estrutura de máquinas de estados finitos de um sistema SDL. Tomando-se esta estrutura como base, é possível aplicar algoritmos utilizando a recursividade e unificação de variáveis, de forma a produzir diagnósticos sobre a semântica do sistema, através do mecanismo de inferência já embutido em PROLOG.

5.1 Um Subconjunto Transformável de SDL

De forma a possibilitar a definição de uma transformação de SDL para PROLOG e tornar factível a análise e verificação de protocolos, abrangendo a solução de classes de problemas notamente importantes, define-se um subconjunto de SDL com os seguintes itens:

- a construção PROCESS;
- a construção STATE;

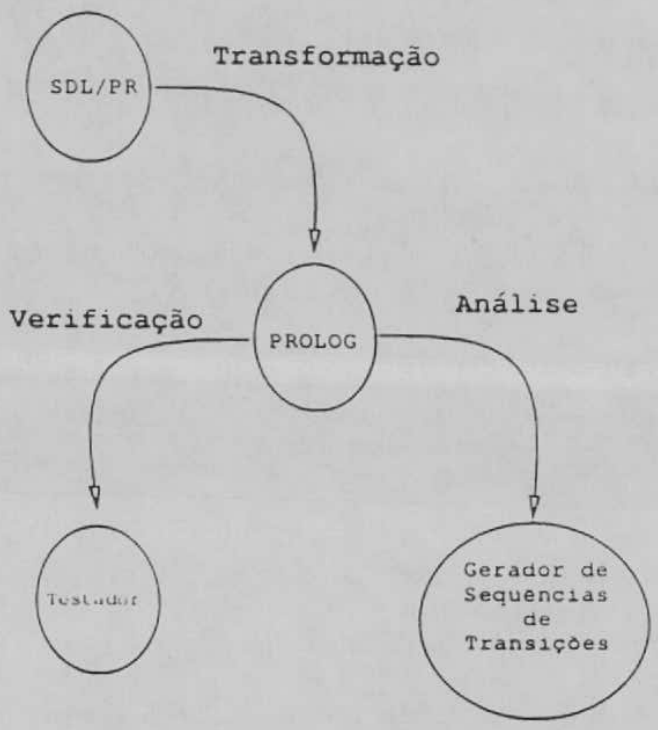

Figura 2: A Arquitetura da Ferramenta

- a contrução INPUT;
- a contrução OUTPUT;

Tais construções são transformadas em equivalente PROLOG, possibilitando a análise e verificação de propriedades desejáveis em protocolos.

5.2 O Ambiente da Implementação

O software utilizado é o Turbo Prolog [BOR 86], um compilador interpretador Prolog. São providos diversos predicados padrão, para entrada e saída assim como comandos especiais para controle da recursividade.

Um sistema de janelas permite que se possa editar, compilar, executar e depurar programas, o que aumenta sobremaneira a produtividade. No ambiente da implementação é possível utilizar também janelas DOS, de forma a examinar prontamente os resultados (saídas) gerados em arquivos, graças ao gerenciador de programas do Windows [MIC 92].

A versão do Turbo Prolog utilizada é relativamente antiga mas foi possível obter uma boa performance na execução dos algoritmos, em um PC 386-DX.

Na figura 4, pode-se visualizar uma janela do ambiente da implementação. Neste exemplo, o interpretador Prolog busca as possíveis soluções para a análise das seqüências de transições da fase conexão do protocolo TCP (programa 8). O número de soluções encontradas para o problema é apresentado após sua execução (104).

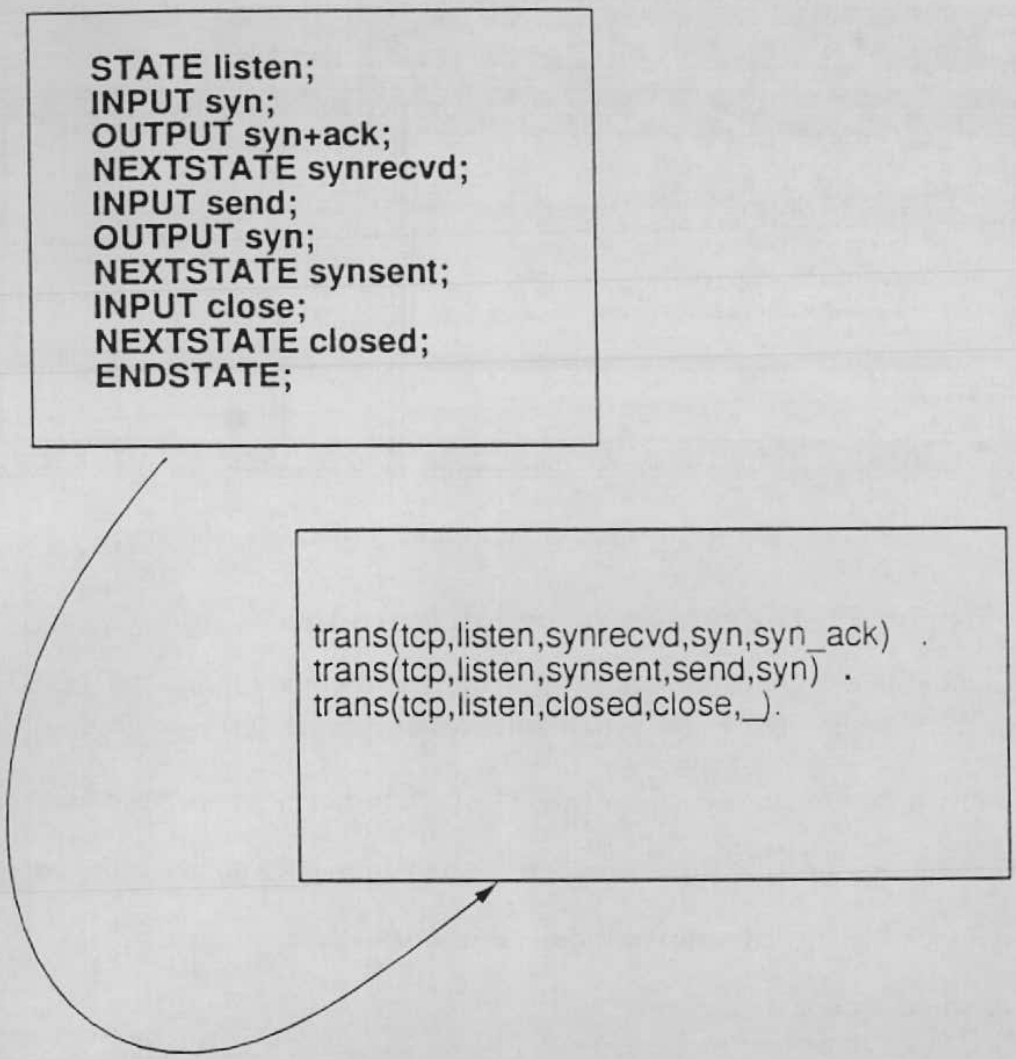

Figura 3: A Transformação de SDL para PROLOG

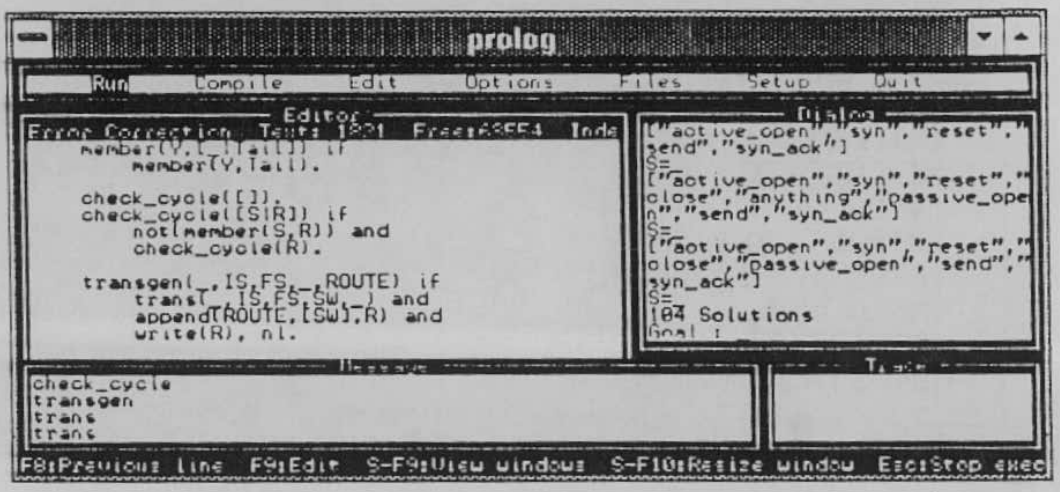

Figura 4: O Ambiente da Implementação

5.3 Erros Detectáveis pela Ferramenta

Através da aplicação de algoritmos (implementados como regras PROLOG) sobre um sistema PSDL, é possível obter atualmente resultados na detecção dos seguintes erros:

- o envio de um sinal perdido (não é esperado por nenhum processo);
- a espera de um sinal que nunca virá (não é enviado por nenhum processo);
- livelocks (ciclos com possibilidade de saída);
- deadlocks do sistema.

A detecção de sinais perdidos ou sinais que nunca alcançarão seu destino é feita por regras simples, que varrem as listas de sinais testando a integridade das transições (figuras 5 e 6). Os sinais são modelados como fatos PROLOG conforme exemplificado na figura 3.

O predicado *notwaited* funciona analogamente ao predicado *notsend*, testando um sinal que é enviado mas não esperado.

O predicado *notsend* é verdadeiro se um sinal pode ser instanciado em uma dada transição como sinal de entrada mas não é instanciado em nenhuma transição como sendo sinal de saída.

Para a detecção de livelocks, são utilizados predicados que realizam a inspeção de ciclos entre as possíveis combinações entre estados.

O programa utilizado para detecção de deadlocks utiliza os mesmos predicados no teste de livelocks, apenas testando se não há rotas alternativas que possam evitar tal situação.

```
notwaited(SIG) if
    trans(_,_,_,_,SIG) and
    not(trans(_,_,_,SIG,_)).

signal_loss([]).
signal_loss([SIG|R]) if
    not(notwaited(SIG)) and
    signal_loss(R).

check_signal_loss if
    signal(LSIG) and
    signal_loss(LSIG).
check_signal_loss if
    write("Signal loss found"), nl.
```

Figura 5: Inspeção de Sinais Perdidos

```
notsend(SIG) if
    trans(_,_,_,SIG,_) and
    not(trans(_,_,_,_,SIG)).

signal_never_come([]).
signal_never_come([SIG|R]) if
    not(notsend(SIG)) and
    signal_never_come(R).

check_signal_never_come if
    signal(LSIG) and
    signal_never_come(LSIG).
check_signal_never_come if
    write("Signal will never come"), nl.
```

Figura 6: Inspeção de Sinais nunca Enviados

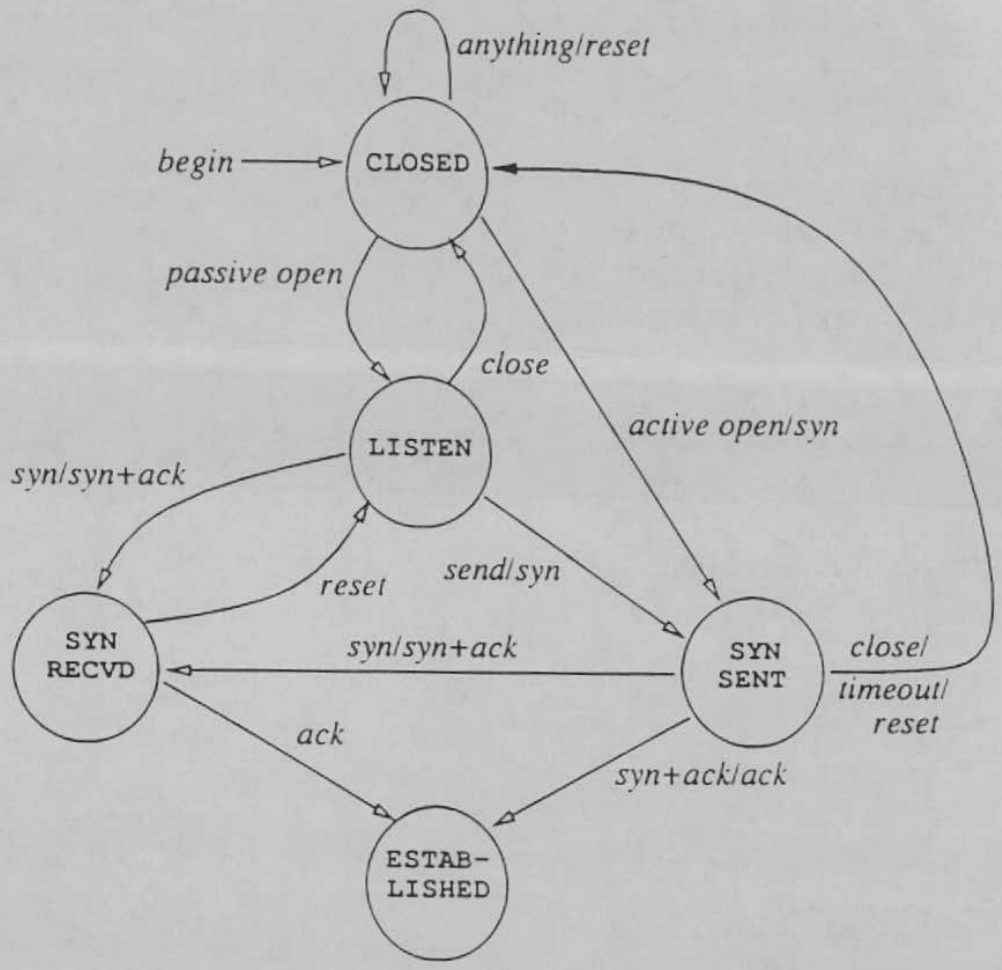

Figura 7: O Estabelecimento da Conexão TCP

6 Análise do Protocolo TCP

De forma a permitir uma aproximação com a análise de um protocolo real, foram abordados os principais aspectos do funcionamento da máquina de estados do protocolo TCP [COM 91]. Para testar o uso da metodologia proposta, foi definida a análise das seqüências de funcionamento do protocolo para a fase de estabelecimento de conexão.

A especificação do subconjunto (máquina de estados) proposto do protocolo TCP pode ser vista na figura 7.

Através da geração das possíveis seqüências de transição entre estados, determinadas pelo recebimento de sinais de entrada, pode-se ter uma idéia do comportamento do protocolo TCP até a fase de conexão. A figura 9 mostra um conjunto de transições geradas com as possíveis seqüências entre os estados *closed* e *established*, sem repetições entre estados.

O predicado utilizado é o *transgen*, cuja implementação pode ser vista na figura 8. Os predicados *append*, *member* e *check_cycle* são os mesmos utilizados para detecção de livelocks e deadlocks, apenas aplicados sobre sinais ao invés de estados.

O predicado *check_cycle* é utilizado para evitar que o programa da figura 8 entre em um laço infinito, devido ao mecanismo de busca de soluções da linguagem PROLOG. Um predicado especial *cut* (!) é utilizado para controlar a busca de soluções para o problema proposto. A sua presença determina que não serão procuradas todas as soluções para a solução de um predicado dentro de uma regra, mas apenas uma, a que primeiramente for instanciada.

O conjunto de todas as possíveis seqüências incluindo repetições na passagem entre os estados não será mostrado, uma vez que que o conjunto das possíveis combinações é bastante extenso, ainda que apenas a fase de conexão do protocolo TCP esteja sendo analisada. O uso do *cut* é imprescindível para controlar a busca de soluções, inclusive para evitar overflow da pilha na execução do programa, devido à característica recursiva de PROLOG.

O predicado *transgen* pode ser utilizado para gerar as possíveis combinações das transições entre dois estados quaisquer. Neste exemplo, pode-se notar que todas as seqüências geradas iniciam pelo estado *closed* e terminam por *established*. Esta divisão é meramente por limitações de espaço, uma vez que o número total de soluções para a máquina de estados TCP completa é significativo (1035).

7 Discussão Comparativa dos Resultados

Nesta seção são discutidos os resultados obtidos e comparados com outras ferramentas que implementam a análise e/ou verificação de especificações SDL.

Atualmente, os diagnósticos produzidos na verificação limitam-se a detectar ou não o erro em questão. Tais diagnósticos poderiam ser facilmente acompanhados de relatórios mais detalhados, bastando para tanto apenas algumas alterações nos programas descritos.

[CAR 89] utiliza a transformação de sistemas SDL para PROLOG, sendo possível detectar alguns erros como deadlocks, sinais enviados e não recebidos, atualização ou leitura de variáveis não inicializadas. É utilizado um escalonador (*scheduler*) que implementa um ambiente para a execução simulada do sistema SDL descrito em PROLOG.

A ferramenta descrita em [KAR 89] usa sessões de simulação de transições, objetivando um entendimento e depuração do comportamento de sistemas SDL, durante a validação de uma especificação, por exemplo. A arquitetura descrita engloba a geração de código Pascal a ser executado simbolicamente, simulando a execução paralela de transições em processos SDL.

Em [NGU 89], é utilizada a técnica de análise de alcançabilidade, cuja idéia geral é baseada na geração exaustiva de todas as possíveis interações entre processos de um sistema. A partir de um dado estado inicial são exploradas todas as possíveis

```
        check_cycle([]).
        check_cycle([S|R]) if
            not(member(S,R)) and
            check_cycle(R).

        transgen(_,IS,FS,_,_,ROUTE) if
            trans(_,IS,FS,SW,_) and
            append(ROUTE,[IS,SW,FS],R) and
            write(R), nl and !.

        transgen(_,IS,FS,_,TESTROUTE,ROUTE) if
            trans(_,IS,INTS,SW,_) and
            append(TESTROUTE,[SW],R) and
            check_cycle(R) and
            append(ROUTE,[IS,SW],R1) and
            transgen(_,INTS,FS,_,R,R1).

    state([closed,listen,synsent,synrecvd]).
    trans(tcp1,closed,closed,anything,reset).
    trans(tcp1,closed,listen,passive_open,_).
    trans(tcp1,closed,synsent,active_open,syn).
    trans(tcp1,listen,synrecvd,syn,syn_ack).
    trans(tcp1,listen,synsent,send,syn).
    trans(tcp1,listen,closed,close,_).
    trans(tcp1,synsent,synrecvd,syn,syn_ack).
    trans(tcp1,synsent,established,syn_ack,_).
    trans(tcp1,synsent,closed,close,_).
    trans(tcp1,synsent,closed,timeout,reset).
    trans(tcp1,synsent,synrecvd,syn,syn_ack).
    trans(tcp1,synrecvd,listen,reset,_).
    trans(tcp1,synrecvd,established,ack,_).

goal
    openwrite(destination,"mydata.tst"),
    writedevice(destination),
    transgen(_,closed,established,S,[],[]),
    fail,
    writedevice(screen).
```

Figura 8: Implementação do Predicado *transgen*

```
[closed,anything,closed,passive_open,listen,syn,synrecvd,ack,
established]
[closed,anything,closed,passive_open,listen,send,synsent,syn_ack,
established]
[closed,anything,closed,passive_open,listen,close,closed,
active_open,synsent,syn_ack,established]
[closed,anything,closed,active_open,synsent,syn_ack,established]
[closed,passive_open,listen,syn,synrecvd,ack,established]
[closed,passive_open,listen,send,synsent,syn_ack,established]
[closed,passive_open,listen,close,closed,anything,closed,
active_open,synsent,syn_ack,established]
[closed,passive_open,listen,close,closed,active_open,synsent,
syn_ack,established]
[closed,active_open,synsent,syn_ack,established]
```

Figura 9: Seqüências de Conexão TCP

transições. Isto é repetido para todos os estados até que nenhum novo estado possa ser alcançado. Os principais erros detectados por esta ferramenta são transições não executáveis, livelocks e deadlocks. O protótipo descrito apresenta cerca de 2000 linhas de código C.

No trabalho descrito em [ZOR 89] podem ser verificadas algumas propriedades semânticas tais como estados que não podem ser alcançados, uma transição sem um estado correspondente ou ausência do símbolo de START. A ênfase desta ferramenta está no desenvolvimento de um protótipo de editor gráfico com inspeção sintática. O número de propriedades semânticas inspecionado é reduzido.

Ferramenta	Código Gerado	Verificação	Análise
[CAR 89]	PROLOG	sim	não
[KAR 89]	PASCAL	não	sim
[NGU 89]	C	sim	sim
[ZOR 89]	C	sim	não

Tabela 1: Comparativo entre Ferramentas para Análise e Verificaccão

O protótipo desenvolvido neste trabalho contempla tanto a análise como a verificação de propriedades de protocolos, sendo utilizada uma transformação para código PROLOG. A ferramenta descrita em [NGU 89] utiliza a geração de código C para implementar tais funcionalidades, tendo sido desenvolvidas cerca de 2000 linhas, em contraste com cerca de 20 regras PROLOG implementadas nesta dissertação.

Assim, a utilização de PROLOG para a análise e verificação de protocolos especificados em SDL propicia um ambiente bastante produtivo, ainda que a geração de código PROLOG esteja distante da implementação real de um protocolo.

8 Considerações Finais

Atualmente, os diagnósticos produzidos na verificação limitam-se a detectar ou não o erro em questão. Tais diagnósticos poderiam ser facilmente acompanhados de relatórios mais detalhados, bastando para tanto apenas algumas alterações nos programas descritos.

Este trabalho apresentou uma metodologia de análise e verificação de protocolos através da transformação de SDL para PROLOG. A técnica proposta envolve três funções principais: a tradução de PROLOG (implementando um máquina abstrata do tipo FSM), a verificação com detecção de erros através da aplicação dos algoritmos de análise do código PROLOG resultante, e a análise das transições entre estados, tendo sido utilizada a fase de conexão do protocolo TCP como exemplo para a validação do projeto.

O uso de verificação proposto através de regras e da máquina de inferência PROLOG consubstancia uma metodologia para o processo de engenharia de software de projetos de sistemas em tempo real, oferecendo um ambiente para inspeção de consistência e análise estrutural de produtos de software, notadamente protocolos.

Propriedades sintáticas e semânticas são verificadas pela ferramenta através da aplicação de regras que garantem a ausência de determinadas classes de erros, tais como deadlocks e livelocks.

O protocolo TCP, padrão de fato na interconexão de sistemas, foi analisado sendo explorado o conjunto de estados da fase de conexão (a fase inicial antes da efetiva transferência de dados). Uma vez que a análise completa de todas as possíveis seqüências de transições entre estados geralmente não é possível, um dos maiores problemas neste tipo de análise é reduzir o número de possíveis combinações. Neste sentido, foi definido um mecanismo de controle de busca de soluções.

O tempo de processamento obtido é encorajador, sendo necessários menos de 30 segundos para a geração de todas as possíveis seqüências de transições do protocolo TCP. Acredita-se que o sistema seja eficiente mesmo para um número elevado de estados.

Referências

[BOR 86] BORLAND International. **Turbo Prolog. Owner's Handbook.** Scotts Valley, CA, 1986. 223 p.

[BOC 84] BOCHMANN, G. **Use of Formal Specifications for Protocol Design, Implementation and Testing.** Protocol Specification, Testing and Validation, 4, 11-14 Jun. 1984. Pennsylvania. **Proceedings.** Amsterdam: North-Holland, 1984. pp. 137-144.

[CAR 89] CARL, D. **The Use of SDL in an ISDN Terminal Design.** SDL Forum, 4, 9-13 Oct. 1989.**Proceedings.** Amsterdam: North-Holland,1989. pp. 367-76.

[COM 91] COMER, D. **Internetworking with TCP/IP: Principles, Protocols, and Architeture**. Volume 1. Seg. Edição. Prentice-Hall, Englewood Cliffs, NJ, 1991. pp. 174-175.

[ISO 81] ISO. **International Standards Organization: Work Program on Formal Description Technics**. ISO/TC97/SC16 N751 rev.

[KAR 89] KARLSON, J.; EK, A. **SSI - An SDL Simulation Tool**. SDL Forum, 4, 9-13 Oct. 1989.**Proceedings**. Amsterdam: North-Holland,1989. pp. 211-18.

[KOY 87] KOYANAGI, K.; MARYANA, K. **Declarative Specifications in PROLOG and SDL**. SDL Forum, 3, 3-10 Apr. 1987.**Proceedings**. Amsterdam: North-Holland,1987. pp. 427-437.

[NGU 89] NGUYEN, H,; JACKSON, L.; PARKER, K. **Reachability Graph Generator for SDL**. SDL Forum, 4, 9-13 Oct. 1989.**Proceedings**. Amsterdam: North-Holland,1989. pp. 219-30.

[MIC 92] MICROSOFT Corporation. **Microsoft Windows**. Guia do Usuário. 1992. 711 p.

[ORA 88] ORAVA, F. **Formal Semantics of SDL Specifications** Protocol Specification, Testing and Validation, 8, 11-14 Jun. 1988. **Proceedings**. Amsterdam: North-Holland, 1988.

[SAJ 84] SAJKOWSKI, M. **Protocol Verification Techniques. Status Quo and Perspectives**. Protocol Specification, Testing and Validation, 4, 11-14 Jun. 1984. Pennsylvania. **Proceedings**. Amsterdam: North-Holland, 1984. pp. 697-720.

[TRI 91] TRINDADE, R. S. **Estudo Comparativo das Linguagens de Especificação e Teste de Protocolos ESTELLE, LOTOS, SDL e CHILL**. Porto Alegre: CPGCC da UFRGS, 1991. 25 p.

[TRI 92] TRINDADE, R. S. **Um Estudo da Linguagem SDL para Especificação e Teste de Protocolos**. Porto Alegre: CPGCC da UFRGS, 1992. 87 p. (Trabalho Individual).

[UNR 87] UNRUH, E.**SCAN, an Expert System for the Analysis of SDL Specifications**. SDL Forum, 3, 3-10 Apr. 1987.**Proceedings**. Amsterdam: North-Holland,1987. pp. 137-146.

[ZOR 89] ZORIĆ, M; GALOVIĆ, Z.; STANTIĆ, G. **Tool Set Development and the use of SDL**. SDL Forum, 4, 9-13 Oct. 1989.**Proceedings**. Amsterdam: North-Holland,

[Z.100] CCITT. **Recommendation Z100: Specification and Description Language**. Genebra: CCITT, 1988. 180 p. (Blue Book, v.10 n.2)