A

Teste de Conformidade em Contexto Guiado por Casos de Teste do Componente

Este exemplar corresponde à redação da Dissertação apresentada para a Banca Examinadora antes da defesa da Dissertação.

Este exemplar corresponde à redação final da Tese/Dissertação devidamente corrigida e defendida por: Sidnei Martins Soares Junior
e aprovada pela Banca Examinadora.
Campinas, 28 de 12 de 07
COORDENADOR DE PÓS-GRADUAÇÃO
CPG-IC

Campinas, 4 de dezembro de 2006.

Prof. Dr. Ricardo de Oliveira Anido
Instituto de Computação – UNICAMP
(orientador)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Instituto de Computação
Universidade Estadual de Campinas

Teste de Conformidade em Contexto Guiado por Casos de Teste do Componente

Jurandy Martins Soares Jr.

4 de dezembro de 2006

Banca Examinadora:

- Prof. Dr. Ricardo de Oliveira Anido
  Instituto de Computação – UNICAMP (orientador)
- Dra. Ana Maria Ambrósio
  Desenvolvimento de Sistemas de Solo – INPE
- Profa. Dra. Eliane Martins
  Instituto de Computação – UNICAMP
- Prof. Dr. Célio Cardoso Guimarães (suplente)
  Instituto de Computação – UNICAMP

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Bibliotecária: Maria Júlia Milani Rodrigues – CRB8a / 2116

Soares Júnior, Jurandy Martins

So11t Teste de conformidade em contexto guiado por casos de teste do componente / Jurandy Martins Soares Júnior – Campinas, [S.P. :s.n.], 2006..

Orientador : Ricardo de Oliveira Anido

Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Redes de computadores – Protocolos. 2. Métodos formais (Ciência da computação). 3. Engenharia de software. I. Anido, Ricardo de Oliveira. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

(mjmr/imecc)

Título em inglês: Conformance testing in context guided by component´s test cases.

Palavras-chave em inglês (Keywords): 1. Computer network – Protocols. 2. Formal methods (Computer science). 3. Software engineering.

Área de concentração: Engenharia de protocolos

Titulação: Mestre em Ciência da Computação

Banca examinadora: Prof. Dr. Ricardo de Oliveira Anido (IC-UNICAMP)
Profa. Dra. Eliana Martins (IC-UNICAMP)
Dra. Ana Maria Ambrósio (DSS-INPE)

Data da defesa: 04/12/2006

Programa de Pós-Graduação: Mestrado em Ciência da Computação

TERMO DE APROVAÇÃO

Tese defendida e aprovada em 04 de dezembro de 2006, pela Banca examinadora composta pelos Professores Doutores:

Profa. Dra. Ana Maria Ambrósio
DSS – INPE.

Profa. Dra. Eliane Martins
IC – UNICAMP.

Prof. Dr. Ricardo de Oliveira Anido
IC – UNICAMP.

Pensamentos

"*Le premier était de ne recevoir jamais aucune chose pour vraie, que je ne la connusse évidemment être telle: c'est-à-dire, d'éviter soigneusement la précipitation et la prévention; et de ne comprendre rien de plus en mes jugements, que ce qui se présenterait si clairement et si distinctement à mon esprit, que je n'eusse aucune occasion de le mettre en doute. Le second, de diviser chacune des difficultés que j'examinerais, en autant de parcelles qu'il se pourrait, et qu'il serait requis pour les mieux résoudre. Le troisième, de conduire par ordre mes pensées, en commençant par les objets les plus simples et les plus aisés à connaître, pour monter peu à peu, comme par degrés, jusques à la connaissance des plus composés; et supposant même de l'ordre entre ceux qui ne se précèdent point naturellement les uns les autres. Et le dernier, de faire partout des dénombrements si entiers, et des revues si générales, que je fusse assuré de ne rien omettre.*"

René DESCARTES (1637). **Discours de la méthode: pour bien conduire sa raison, et chercher la vérité dans les sciences.**

"*Some may argue that one could achieve this goal just by building theoretical models and running computational simulations. Perhaps that is true. But as my good friend Idan Segev, a leading computational neuroscientist, always tells me, there is a subtle but fundamental difference between simulating reality and building it. Those of us who saw Pelé scoring that magic goal on that hot Mexican afternoon in 1970 and dreamed about doing the same thing would certainly agree*."

Miguel A. L. NICOLELIS (2001). **Actions from thoughts**. Nature 409: 403-407

Agradecimentos

Aos meus pais, Jurandy e Emília, pelos esforços que fizeram em formar humildemente e honestamente todos os sete filhos. Aos irmãos Maninha, Dedé, Jesus, Kátia, Layna e Emília pelo apoio que sempre me deram. Aos sobrinhos e quase irmãos Denise e Guilherme, e aos demais sobrinhos Jéssica e Jéslley. À minha esposa e companheira dos últimos anos, Adriana Kaysa. Às amizades criadas e cultivadas ao longo desta jornada em Campinas, em Teresina, em Brasília e em Paris.

Agradeço em especial ao professor Ricardo Anido, que me apresentou o domínio sob o qual tenho trabalhado atualmente. E também à oportunidade que me foi dada para viajar à França, onde muito aprendi sobre o que existe na área. Ao pessoal de Evry, em especial à professora Ana Cavalli, pela paciência e sapiência. Aos amigos e colegas de diferentes nacionalidades que conheci em Evry e em Paris, com quem aprendi a importância de protocolos de comunicação, porque a linguagem humana é, de uma forma ou de outra, um protocolo de comunicação muito mais complexo do que os protocolos de comunicação implementados em equipamentos eletrônicos.

Agradeço também aos colegas investigadores, aos colegas alunos, aos colegas de trabalho e aos professores do Instituto de Computação da Unicamp. Agradeço em especial ao colega André Vasquez, cujos experimentos em conjunto foram vitais para a escrita do Capítulo 6. A todos citados anteriormente, e aos demais que por um lapso de memória não os citei, os meus sinceros agradecimentos.

Por fim agradeço às doutoras presentes em minha banca de defesa de mestrado – professora Eliane Martins do Instituto de Computação da Unicamp e pesquisadora Ana Ambrósio do Departamento de Desenvolvimento de Sistema de Solo do Instituto Nacional de Pesquisas Espaciais (DSS-Inpe) – pelos comentários finais que deixaram esta dissertação mais compreensível e estruturada.

Resumo

Testar um subsistema embarcado em um sistema complexo, assumindo-se que os demais subsistemas são livres de falhas, é conhecido como teste de conformidade em contexto. A complexidade deste teste reside no fato do subsistema mais externo, conhecido como contexto, ocultar muitas interações e eventos nos quais o sistema embarcado, conhecido como componente, participa. Nas últimas décadas alguns algoritmos foram desenvolvidos para resolver o problema. Muitos deles, no entanto, ignoram as condições nas quais podem ser aplicados. Nesta dissertação estudamos a teoria e os algoritmos relacionados a teste de conformidade e a teste de conformidade em contexto, propomos uma nova abordagem guiada por casos de teste do componente e analisamos as condições necessárias e suficientes para aplicá-la. A validação das condições necessárias e suficientes foi feita em estudos de casos com a pilha de protocolos do TCP/IP, com os protocolos HTTP e HTTPS via Proxy e com DHCP via *relay-agent*. Os algoritmos estudados foram experimentados nas especificações dos protocolos ABP, SCU e TCP.

Abstract

Testing a subsystem embedded into a complex system, in which the other subsystems are assumed to be fault-free, is known as conformance testing in context. The complexity behind this test resides in the fact that the most external system, known as context, hides many interactions and events in which the embedded subsystem, known as component, participates. In the last decades some algorithms were developed to solve the problem. Many of them, however, ignore the conditions under which they can be applied. In this dissertation we study the theory and the algorithms related to conformance testing and conformance testing in context, we propose a new approach guided by component's test cases, and we analyze the necessary and sufficient conditions to apply it. The validation of the necessary and sufficient conditions was done in case studies with the TCP/IP protocol stack, with the protocols HTTP and HTTPS via Proxy, and with DHCP via relay-agent. The algorithms studied were experimented in the specifications of protocols ABP, SCU e TCP.

Lista de Acrônimos

ABP	Alternating Bit Protocol
DHCP	Dynamic Host Configuration Protocol
EFSM	Extended Finite State Machine
FDT	Formal Description Techniques
FSM	Finite State Machine
HTTP	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Secure Hypertext Transfer Protocol
IAP	Implementation Access Point
IEEE	Institute of Electrical and Electronics Engineers
INPE	Instituto Nacional de Pesquisas Espaciais
ISO	International Organization for Standardization
IST	Implementação Sob Teste
IUT	Implementation Under Test
OSI	Open Systems Interconnection
PCI	Protocol Control Information
PCO	Point of Control and Observation
PDU	Protocol Data Unit
SAP	Service Access Point
SDL	Specification Description Language
SDU	Service Data Unit
SSL	Security Socket Layer
TCP	Transmission Control Protocol
WAP	Wireless Application Protocol
WTLS	Wireless Transport Layer Security

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Teste em Contexto: a avaliação de um subsistema a partir de outro

1.1 Introdução

A Figura 1.1 mostra a pilha de protocolos do TCP/IP [Socolofsky and Kale, 1991] em que subsistemas de uma camada comunicam-se com subsistemas de camadas adjacentes. Esta comunicação só é possível porque antes da implementação dos protocolos de cada camada uma interface de serviços foi especificada entre elas.

Esta observação, assim como outras relacionadas à engenharia de protocolos, embora possam parecer triviais, suscita a questão sob a qual o estudo científico de teste de conformidade de protocolos em contexto é baseado: Podemos avaliar um subsistema por meio de outro mais externo? [Lee et al., 1993]

Esta questão cobre até certo ponto um conjunto complexo de problemas. Um deles é determinar e representar a estrutura causal do comportamento de cada um dos subsistemas [Mealy, 1955, Moore, 1956, Petri, 1962, Hoare, 1985, Milner, 1989, Lynch and Tuttle, 1989]. Como veremos, neste caso particular um fator determinante é o estímulo exercido pelo ambiente. O teste simples demonstrado em capítulos posteriores é, a saber, solicitar um documento Web por meio de um cliente HTTP (Protocolo de Transferência de HiperTexto[1].) [Fielding et al., 1999], o que desencadeia uma seqüência de interações entre os protocolos de diferentes camadas. A representação do comportamento de cada subsistema é feita principalmente por meio de máquinas de estados.

Outro fator causal é de natureza interna. O fato de uma comunicação HTTP não ser

[1] do Inglês *HyperText Transfer Protocol*

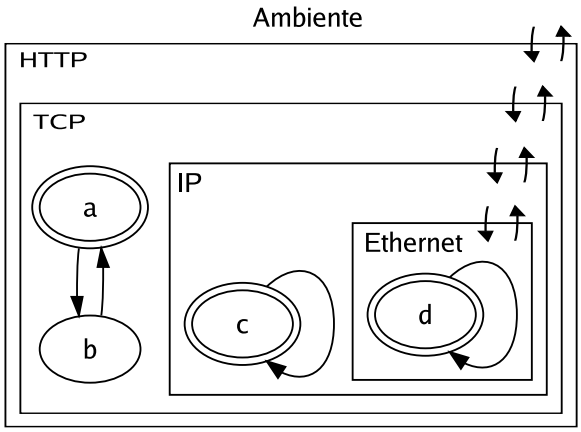


Figura 1.1: Abstração da pilha de protocolos do TCP/IP com HTTP

bem sucedida pode ser decorrência do servidor de nomes (DNS) da máquina que executa o cliente HTTP está indisponível, dentre outras causas relacionadas a protocolos de camadas inferiores. Isto sugere que sinais internos influenciam os estados dos subsistemas embutidos. Se isso pudesse ser provado por experimentos, representaria um outro passo na direção de uma solução para teste de conformidade em contexto.

Ambos os sinais ordinários internos e externos exercem suas influências no subsistema embarcado por meio de canais. Portanto, nossa próxima tarefa após o estudo dos estímulos externos e eventos internos envolvidos teria que ser como avaliar individualmente cada um dos subsistemas, abstraindo que podemos controlar e observar o que entra e o que sai em cada um deles. Quando fazemos uma descrição acurada do tipo particular de comportamento mostrado no presente caso, devemos ver que muitos subsistemas estão envolvidos e cada um (1) manipula um tipo de dado específico e (2) em coordenação com os outros subsistemas. Isto indica a complexidade dos processos envolvidos nesta simples ação.

Assim, o problema da avaliação do comportamento de um subsistema embutido leva ao estudo da representação e avaliação individual de cada um dos subsistemas envolvidos, de métodos para testar implementações diferentes de um mesmo subsistema, de máquinas de estados e, particularmente, da coordenação entre os subsistemas e sua integração no ato do comportamento do sistema como um todo.

Entretanto, se resolvêssemos este problema particular – isto é, se descobríssemos uma maneira eficiente de testar o subsistema embutido – teríamos ainda assim respondido apenas parte da questão. Pelo fato de um subsistema ser resultado do processo de desenvolvimento que começou com sua especificação, a descrição e explicação do processo de desenvolvimento de um protocolo são extensões deste estudo. Em outras palavras, é parte de nossa tarefa estudar

maneiras de se desenvolver e representar protocolos.

Mesmo esta extensão da engenharia de protocolos não cobre o campo completo de teste de conformidade em contexto. Também desejamos saber em que condições é possível avaliar um subsistema embutido através de outro mais externo. Ao contrário do que comumente se encontra na literatura, pode ser que não seja possível, ou bastante limitado. Assim, querendo ou não, somos conduzidos ao estudo da evolução dos algoritmos desenvolvidos ao longo do tempo. Corroborando com as dificuldades especiais que encontramos ao desenvolver um novo algoritmo, existem condições bastante específicas em que se pode aplicar teste a um subsistema embutido.

Retornando à nossa questão principal, parece óbvio que ela cobre mais do que o problema das causas de mudança de estado do subsistema embutido. Quando visto por experimento, como será visto mais tarde, a tentativa de avaliar algum subsistema embutido da pilha de protocolos do TCP/IP esbarra-se na limitação que cada subsistema foi desenvolvido para tratar tipos diferentes de dados, dos quais os sinais dependem.

Assim o problema principal divide-se em vários outros mais ou menos separados, cada um matéria de campos separados de ciência da computação. O estudo da representação e especificação de um protocolo está relacionado com métodos formais. O estudo de como implementar o protocolo em uma plataforma e linguagem de programação está ligado à programação de sistemas. O estudo de como testá-lo está diretamente relacionado à área de testes de engenharia de protocolos e de software e a algoritmos de travessia de grafos. O estudo da composição de subsistemas para teste de um interno contém aspectos de desenvolvimento estruturado, de métodos formais e de circuitos digitais. Tudo isso culmina no estudo de coordenação, de integração, entre os subsistemas.

Neste trabalho nos concentraremos em somente parte desse conjunto de problemas, e enquanto obviamente seria indesejável negligenciar algum dos campos mencionados, trataremos principalmente com aspectos formais e algoritmos para teste de conformidade de protocolos em contexto.

1.2 Motivação Tecnológica

Assim como na biosfera temos uma quantidade enorme de seres vivos e o teste do comportamento destes depende do estudo do comportamento individual, em família e em sociedade [Tinbergen, 1989, Chauvin, 1975], na tecnosfera temos cada vez mais uma proliferação de equipamentos eletrônicos de telecomunicação e o teste destes depende do teste individual de um protocolo do equipamento, do teste do comportamento da implementação deste protocolo

frente a outros da mesma família ou pertencentes à mesma camada, frente a outros de camadas distintas, e ao teste do equipamento em comunicação com outros compatíveis existentes na tecnosfera.

A proliferação desses equipamentos pode ser observada pela sua disponibilidade crescente no mercado e sua aceitação por parte dos usuários. Em particular, a proliferação de telefones celulares é bastante ilustrativa, pois estes são desenvolvidos para uma comunicação em duplo sentido (*full-duplex*), comunicam-se entre si diretamente, por meio de uma rádio base ou de várias rádios base e por meio de protocolos de tecnologias distintas. Possuem acesso à Internet e já é comum a venda de modelos que sintonizam ondas de rádio e de televisão. O gráfico da Figura 1.2 ilustra o uso crescente de celulares [Mateus and Loureiro, 1999]. Enquanto a televisão preto e branco levou 20 anos para chegar ao patamar de um milhão de usuários desde o seu lançamento, a televisão colorida levou metade desse tempo, o vídeo cassete levou aproximadamente oito anos, o computador pessoal levou aproximadamente seis anos, o celular analógico levou dois anos e celular com tecnologia totalmente digital levou menos de um ano.

Logo, o tempo desde a concepção até o lançamento no mercado é cada vez menor e a quantidade vendida do produto é cada vez maior. Os testes dos protocolos embutidos nos celulares assumem então grande importância pois é cada vez menor seu tempo de lançamento ao mercado e maior a quantidade produzida, tornando maior o custo de correção de uma eventual falha [Beizer, 1990].

Em poucas palavras podemos dizer que o teste e a descrição formal de protocolos assumem grande importância quando há risco de grandes perdas, quer seja humanas, ambientais, financeiras ou de qualquer outro gênero [Clarke and Wing, 1996, Martins et al., 1999]. Dentre estas áreas encontram-se a indústria aeroespacial, a indústria ferroviária e a indústria de telecomunicações.

1.3 Objetivos, escopo e limitações do estudo de Teste em Contexto

Como nosso propósito é estudar a viabilidade de se efetuar teste em contexto, nosso campo de pesquisa estará limitado a duas questões principais: 1) Pode-se efetuar teste em contexto? 2) Em caso afirmativo, de que modo? Essas duas questões são interdependentes e, tendo respostas positivas, em que medida? São ambas igualmente importantes? Pois, se pudermos mostrar que é possível avaliar o comportamento de um protocolo por meio de outro, deverá haver meios de fazer e medir isso. Ademais é nosso objetivo procurar solução mais adequada ao problema do que as existentes na literatura.

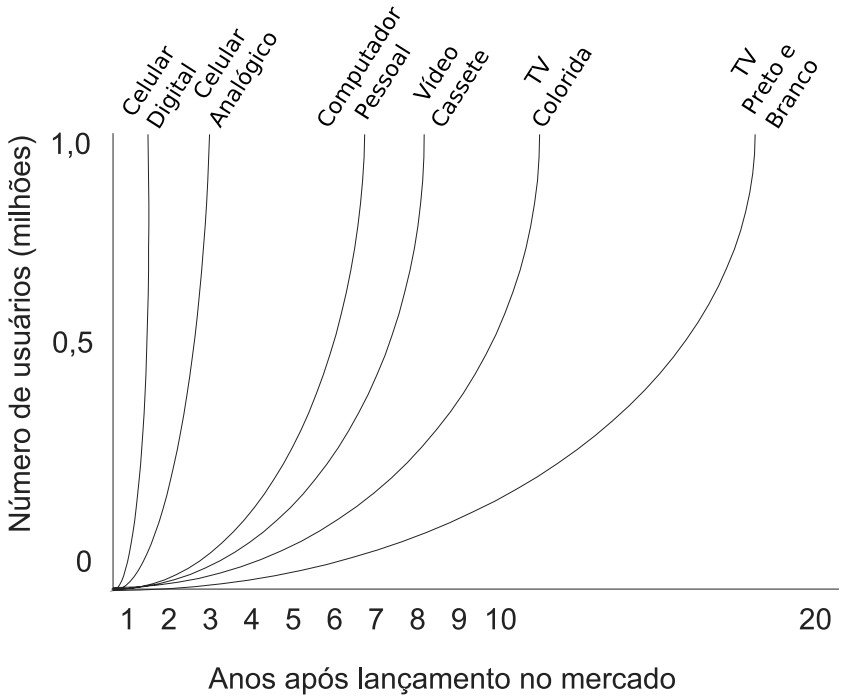


Figura 1.2: Números de usuários por ano de introdução da tecnologia

Antes de responder a essas questões apresentaremos as maneiras mais utilizadas de descrever ou representar formalmente protocolos. Isto será visto no Capítulo 2. Uma vez escolhida uma representação formal de protocolos, serão apresentados algoritmos para este modelo. Concomitantemente serão apresentados critérios de medidas de teste. Considerando-se já o problema de teste em contexto, no Capítulo 3 serão apresentados os algoritmos elementares encontrados na literatura. No Capítulo 4 foi estudada a viabilidade de fazer teste em contexto com protocolos reais e do cotidiano. No Capítulo 5 apresentamos nossa solução. No Capítulo 6 apresentamos resultados de experimentos com os algoritmos em especificações de protocolos colhidas durante o período da pesquisa. Finalmente, no Capítulo 7 fazemos nossas conclusões finais.

Capítulo 2

Teste de conformidade

Neste capítulo é apresentada a representação de protocolos adotada para a pesquisa e são listados os algoritmos e os conceitos básicos relacionados. De início serão mostradas a notação e a terminologia utilizadas para representar máquinas de estados. Segundo, serão introduzidos os conceitos de mensagens de reinicialização e de indicação de estado. Em seguida será feita uma revisão dos algoritmos mais relevantes, como o algoritmo para encontrar um passeio de transições, um para encontrar seqüências de distinção de estados, outro para encontrar seqüências de caracterização de estados e outro para encontrar seqüências únicas de sinais de entrada e saída.

2.1 Representação de Protocolos

Há diferentes maneiras de modelar o comportamento de um protocolo ou um sistema. As principais são: máquinas de estados finitos com sinais de entrada e saída (FSM) [Mealy, 1955] [Moore, 1956][Lynch and Tuttle, 1989][von Bochmann and Sunshine, 1980], redes de Petri [Petri, 1962] e álgebra de processos [Hoare, 1985][Milner, 1989] . Desses o mais utilizado é o modelo de FSM, que também será adotado neste trabalho. O uso de FSM possui como principais vantagens as facilidades de geração de código executável e de geração automática de seqüências de testes. O uso de FSMs é ratificado por muitas linguagens de especificação de protocolos. Dentre as principais destacam-se SDL e Estelle. SDL é a mais utilizada e é suportada por ferramentas comerciais como ObjectGEODE[Verilog, 2002], Cinderella [Cinderella, 2002] e JADE[Pereira et al., 2000]. Uma revisão comparativa dessas modelagens pode ser vista em [Sgroi et al., 2000].

2.2 Terminologia e notação

Definição 1 *Uma máquina de estados finitos de Mealy [Mealy, 1955] é uma sêxtupla* $\langle S, I, O, \delta, \lambda, s_0 \rangle$*, tal que:*

(1) S, I *e* O *são conjuntos finitos, onde:*
S *é um conjunto de estados;*
I *é um conjunto de sinais de entrada, também conhecido como alfabeto de entrada;*
O *é um conjunto de sinais de saída, também conhecido como alfabeto de saída.*

(2) $\delta : S \times I \rightarrow 2^S$ *é a função de transição de estado.*

(3) $\lambda : S \times I \rightarrow 2^O$ *é a função de saída.*

(4) s_0 *é o estado inicial.*

Definição 2 *Uma máquina de estados finitos é dita não determinista quando possuir pelo menos um estado que tenha mais de uma transição que aceite um mesmo sinal de entrada. O teste de conformidade para FSM não determinista é comumente considerado quando a FSM não determinista é observável. Uma FSM não determinista é dita observável quando, para cada estado da FSM, for válido: todo par de transições que possui um mesmo sinal de entrada produz sinais de saída distintos [Boroday et al., 2002].*

Definição 3 *Uma seqüência de entradas e saídas de uma FSM é uma seqüência de sinais de entrada, alternado por sinais de saída. Os sinais de saída são produzidos por uma FSM que está executando uma seqüência finita de transições a partir de uma certa configuração, em resposta aos sinais de entrada.*

Definição 4 *Uma configuração* B_i *é um conjunto de estados do sistema que estão ativos em um passo da computação, sendo que* B_0 *é a configuração inicial. Em cada passo supõe-se que os eventos associados à configuração atual sejam válidos e disparam as transições relacionadas, de modo que seja viável modelar o espaço de configurações possíveis do sistema.*

No caso de teste de conformidade de protocolos em contexto tem-se dois subsistemas representados por duas FSMs não deterministas e observáveis [Lee et al., 1993], T e P, onde:

$$T = \langle S_T, I_T, O_T, \delta_T, \lambda_T, s_0 \rangle \quad \text{é chamado de contexto e}$$
$$P = \langle S_P, I_P, O_P, \delta_P, \lambda_P, r_0 \rangle \quad \text{é chamado de componente.}$$

O ambiente não será considerado uma máquina de estados, mas somente um conjunto não necessariamente finito de entradas e um conjunto não necessariamente finito de saídas, denotado por $A = \langle I_A, O_A \rangle$. A Figura 2.1 ilustra a arquitetura do sistema.

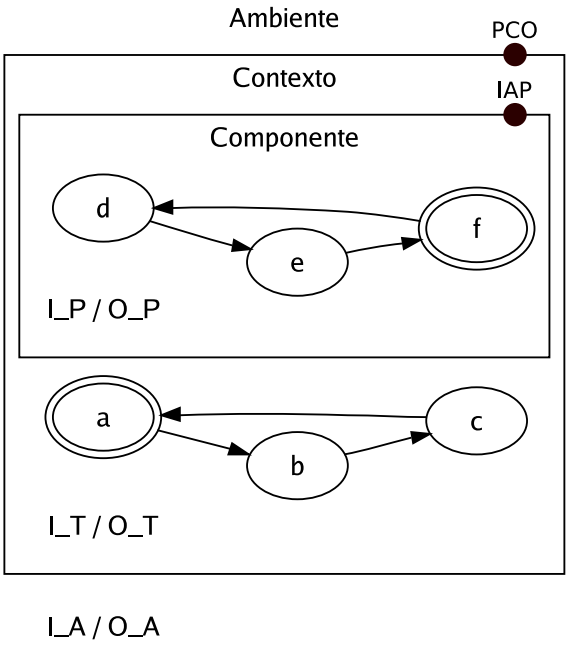


Figura 2.1: Arquitetura para teste em contexto

As propriedades do sistema são:

a) $(I_A \cap O_T \neq \emptyset) e (I_T \cap O_A \neq \emptyset)$, isto é, o conjunto de sinais de entrada do ambiente possui intersecção com o conjunto de sinais de saída do contexto, e vice-versa;

b) $(I_T \cap O_P \neq \emptyset) e (I_P \cap O_T \neq \emptyset)$, isto é, o conjunto de sinais de entrada do contexto possui intersecção com o conjunto de sinais de saída do componente, e vice-versa.

As mensagens trocadas entre o componente e o contexto não são visíveis ao ambiente e são conhecidas como mensagens internas, que contêm sinais internos. Estes são representados pelo conjunto $(I_T \cap O_P) \cup (I_P \cap O_T)$. Conseqüentemente, o conjunto de sinais externos é descrito pelo conjunto $(I_A \cap O_T) \cup (I_T \cap O_A)$. A comunicação entre o ambiente e o contexto se dá por meio de uma interface conhecida como Ponto de Controle e Observação (PCO). O conceito de PCO está definido na metodologia de teste de conformidade ISO-9646 [Consortium, 1994]. A comunicação entre o contexto e o componente se dá por meio de uma interface que será chamada de Interface de Acesso ao Protocolo (IAP).

Existem questões relacionadas a sinais internos e externos que são a controlabilidade e a observabilidade. A controlabilidade refere-se à capacidade que o ambiente tem de interagir e controlar as saídas que serão produzidas (ou que se espera que sejam produzidas) por uma implementação de uma especificação. A observabilidade refere-se à capacidade que o ambiente tem de observar saídas produzidas por uma implementação. No caso de teste em contexto, o contexto é controlável e observável e, o componente, não é controlável e eventualmente poderá ser observável.

A análise formal de protocolos presume, também, que a especificação do protocolo tenha certas propriedades. Estas características são inerentes para todo sistema de aplicações críticas. Estas características consistem de ausência de impasse (*deadlock*), completude, atividade (*liveness*), realização de progresso, terminação, correção parcial, minimalidade e estabilidade.

A ausência de impasse serve para garantir que sob nenhuma condição o protocolo deve atingir um estado de inatividade total e nele permanecer por tempo indefinido. A completude assegura que uma resposta a todas as entradas possíveis foi especificada para cada estado. A atividade assegura a migração do protocolo, de um estado para outro, de forma que se partindo de qualquer estado, todos os demais serão (eventualmente) alcançados. A realização de progresso garante que o protocolo não permanecerá em um estado "inútil", e que alguma "coisa boa" (progresso) irá acontecer [Alpern and Schneider, 1985] . A terminação garante que cada operação do protocolo terminará em tempo finito. A correção parcial garante que ao término de uma operação o protocolo produz o resultado correto. A minimalidade delineia que o protocolo abrange apenas as situações possíveis de ocorrer. E por último a estabilidade, que se certifica que após uma falha o protocolo retorna à operação normal dentro de um tempo finito (ligado à propriedade chamada de "auto-sincronização").

A partir das definições acima, o objetivo é procurar uma seqüência de entradas e saídas, pertencente ao conjunto de sinais externos, que estimule os sinais internos, visando testar da maneira mais completa possível o componente P.

2.3 Critérios de medida de testes

O critério utilizado para avaliar a eficiência de uma seqüência de testes é seu grau de cobertura de falhas. Antes de abordar cobertura de falhas faz-se necessário explicar a diferença entre falha, erro e defeito. *Falha* (do inglês *Fault*) é um evento que causa *erros* (do inglês *Error*). Caso o estado errôneo do sistema não seja tratado em um tempo determinado, ocorrerá um *defeito* (do inglês *Failure*), que se manifestará pela não execução ou mudança indesejada no serviço especificado. Como pode ser visto na Figura 2.2, além de indicar um estado errôneo

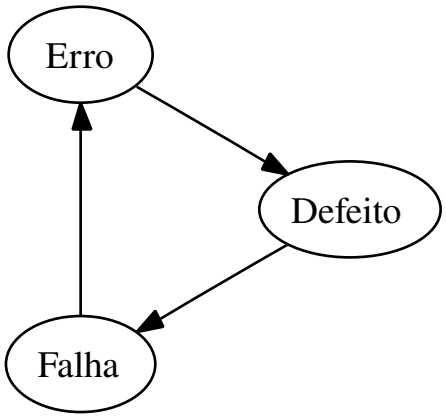


Figura 2.2: Ciclo entre a ocorrência de *Falhas*, *Erros* e *Defeitos*

irreversível, a ocorrência de um defeito realimenta o ciclo e pode gerar novas falhas. Esse fenômeno é conhecido como propagação da falha e deve ser contido através do seu confinamento [Crouch, 1999].

As causas das falhas são diversas e imprevisíveis; elas podem ser conseqüência tanto de fenômenos naturais internos ou externos ao sistema, por exemplo, desgaste de hardware, quanto de ações humanas acidentais ou intencionais, por exemplo, a desconexão de um computador da sua rede interna. Existem várias classificações para falhas na literatura técnica [Anderson and Lee, 1990, Laprie, 1985, Jansch-Pôrto, 1989]. Normalmente, essas classificações agrupam as falhas em falhas físicas, que se referem a falhas de componentes de hardware, e falhas humanas. Falhas humanas compreendem falhas de projeto, decorrentes das fases de desenvolvimento do software e falhas de interação, que por sua vez, podem ser acidentais ou intencionais.

Retornando às definições de falha e defeito, muitos defeitos que acontecem podem ser mapeados em um modelo de falha. Mas um simples defeito, como a implementação de algumas linhas errôneas de código de um protocolo pode não representar uma falha. O termo *falha* possui dois requisitos que devem ser satisfeitos: 1) que o modelo de falha seja exercitável e observável; e 2) que um critério de mensuração de falhas seja estabelecido.

O método de teste convencional para protocolos, também conhecido como método da caixa preta, considera a especificação e a IST (Implementação Sob Teste) como uma FSM. O método de teste assume que ambas, especificação e implementação, possuem o mesmo número de estados ou, excepcionalmente, que o número de estados na implementação seja limitado por uma constante m, onde $m \leq n$ e n é o número de estados da especificação. Para uma seqüência de teste aplicada, se a saída observada na IST for igual à esperada, então é dito que a IST passou pelo teste. Entretanto, se a saída observada diferir da saída esperada, é dito que uma

falha foi encontrada. Os três principais tipos de falhas para o modelo de máquina de estados são: [Neelakantan and Raghavan, 1995]

- **Falha de saída:** Uma IST possui uma falha de saída se, para o estado correspondente e a entrada recebida, a IST produzir uma saída diferente da existente na especificação;
- **Falha de transição:** Uma IST possui uma falha de transição se, para o estado correspondente e a entrada recebida, a IST entrar em um estado diferente, que não aquele descrito na especificação;
- **Falha de transição ausente:** Uma IST tem uma transição ausente se, para o estado atual, não existir uma transição que aceite o próximo sinal de entrada da seqüência de teste.

Os métodos de geração de seqüências de teste mostrados adiante visam cobrir um ou mais desses tipos de falhas. Existem outros tipos de falhas relacionados ao modelo de máquina de estados, mas estes geralmente não são considerados na área de teste de conformidade em engenharia de protocolos. Uma descrição mais detalhada desses tipos pode ser encontrada em [Beizer, 1990, Binder, 1999].

2.4 Mensagens de reinicialização e de indicação de estado

Na área de teste de conformidade de protocolos por vezes é considerada a existência de dois sinais especiais de entrada, que são o sinal de reinicialização e o sinal de indicação de estado. O primeiro serve para conduzir o sistema ao seu estado inicial, i.e., corresponde a um sinal de *reset*. O segundo, como o próprio nome sugere, é um sinal especial que quando enviado provoca um sinal informativo da máquina de estados indicando em que estado ela se encontra. Importante observar que o sinal de *reset*, ao contrário do sinal de indicação de estado, não produz nenhum sinal de saída.

Há variantes desta metodologia como pode ser observado nos *smart cards*, onde existe um modo em que o equipamento permite ser testado, que é o modo padrão quando o equipamento termina de ser fabricado. E existe um outro, que não permite o teste do equipamento. Uma vez neste último, o sistema não mais poderá ser testado, sob pena de danificar o chip do cartão.

O sinal de *reset* é implementado em protocolos reais. Ele pode ser percebido, por exemplo, na especificação do lado cliente do protocolo WTLS (*Wireless Transport Layer Security*), apresentado na Figura 2.3, onde percebe-se nitidamente transições de todos os estados ao estado inicial (**NULL**). O protocolo WTLS é a camada de transporte com segurança das redes de celulares que trabalham com a tecnologia WAP (*Wireless Application Protocol*).

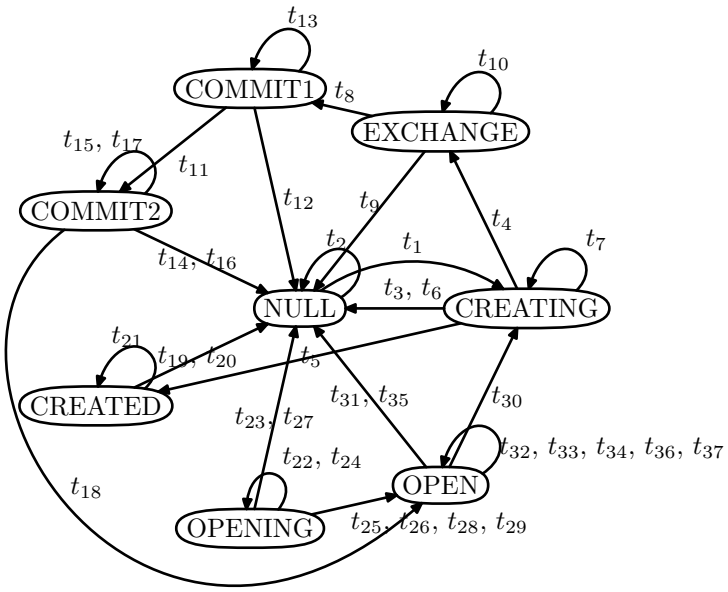


Figura 2.3: Máquina de estados do cliente WTLS (WAP)

2.5 Técnicas de derivação de testes

Uma vez definidos a terminologia adotada para descrever o comportamento dos protocolos, os critérios de medida de teste e os conceitos de mensagem de reinicialização e de indicação de estado, serão vistas nas seções seguintes deste capítulo as principais técnicas de derivação de seqüências de teste utilizadas em teste de conformidade de protocolos.

A primeira técnica a ser vista é o passeio de transições, que serve para encontrar seqüências de teste que cobrem falhas de sinais de saída. As técnicas seguintes geram seqüências de teste que cobrem falhas de transição de estado.

2.5.1 Passeio de transições (método T)

O algoritmo descrito nesta seção é o algoritmo que gera um passeio mínimo de transições ou algum passeio próximo do mínimo. Um passeio de transições[1] é um passeio que percorre todas as transições de uma dada máquina de estados. Este é um problema padrão de teoria dos grafos. Um caso particular de passeio de transições é o passeio de Euler. Um passeio de Euler é uma seqüência de transições que inicia e termina num mesmo estado e que contém uma instância de todas as transições. O passeio de Euler requer que a especificação de um protocolo seja fortemente conexa e simétrica, isto é, todo estado deve ser origem e destino de um mesmo

[1] do Inglês *transition tour*.

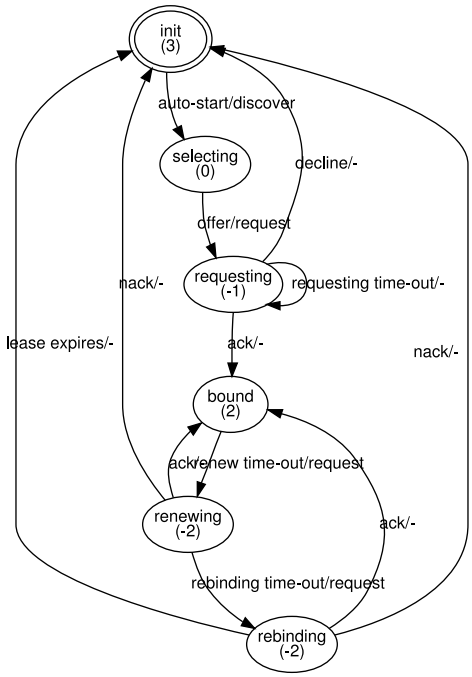


Figura 2.4: Especificação resumida do protocolo DHCP com graus dos estados

número de transições. O algoritmo que gera o passeio de Euler é descrito logo abaixo:

Algoritmo 1 Derivação de um passeio de transições

Seja $SpnArb$ e $TTour$ as variáveis que armazenararão, respectivamente, a arborescência geradora e as seqüências de transições do passeio.

1. **Arborescência Geradora** – Pegue o estado inicial s_0 e adicione-o a $SpnArb$. Este vértice será a raiz da arborescência geradora. Depois selecione uma transição (s, i) com $s \notin SpnArb$ e $dest(s, i) \in S$. Adicione o estado s e a transição (s, i) à arborescência geradora. Continue a aumentar a árvore até que todos os vértices sejam adicionados.

2. **Passeio de transições** – Começando do estado inicial $s = s_0$, selecione uma transição não visitada de s, adicione s e a transição escolhida ao passeio de transições ($TTour$), e então mova para seu estado destino. Agora s assume o valor do estado destino e passo anterior é repetido até que não haja mais transição não visitada no passeio ($TTour$). Uma prioridade menor na seleção é dada para as transições que pertencerem à arborescência geradora. As outras transições podem ser escolhidas em ordem arbitrária. Continue a aumentar o passeio de transições até que nenhuma nova aresta possa ser adicionada.

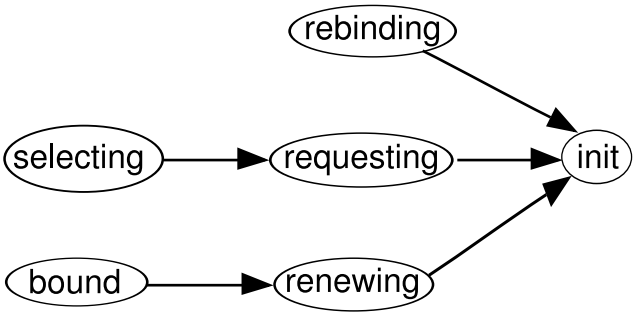


Figura 2.5: Arborescência geradora do protocolo DHCP

Esse algoritmo de passeio de transições é muito simples e eficiente, entretanto, não raro é encontrar especificações de protocolos que não sejam simétricas. Nestes casos primeiramente aplica-se o processo de simetrização, que consiste da duplicação de algumas transições. Os estados que terão algumas de suas transições duplicadas serão aqueles cuja diferença entre as transições que saem e as que chegam seja diferente de zero. A esta diferença dá-se o nome de grau. As transições duplicadas simbolizam que as transições que elas representam serão executadas mais de uma vez no passeio de transições. O Algoritmo 2 descreve todos os passos do procedimento de simetrização.

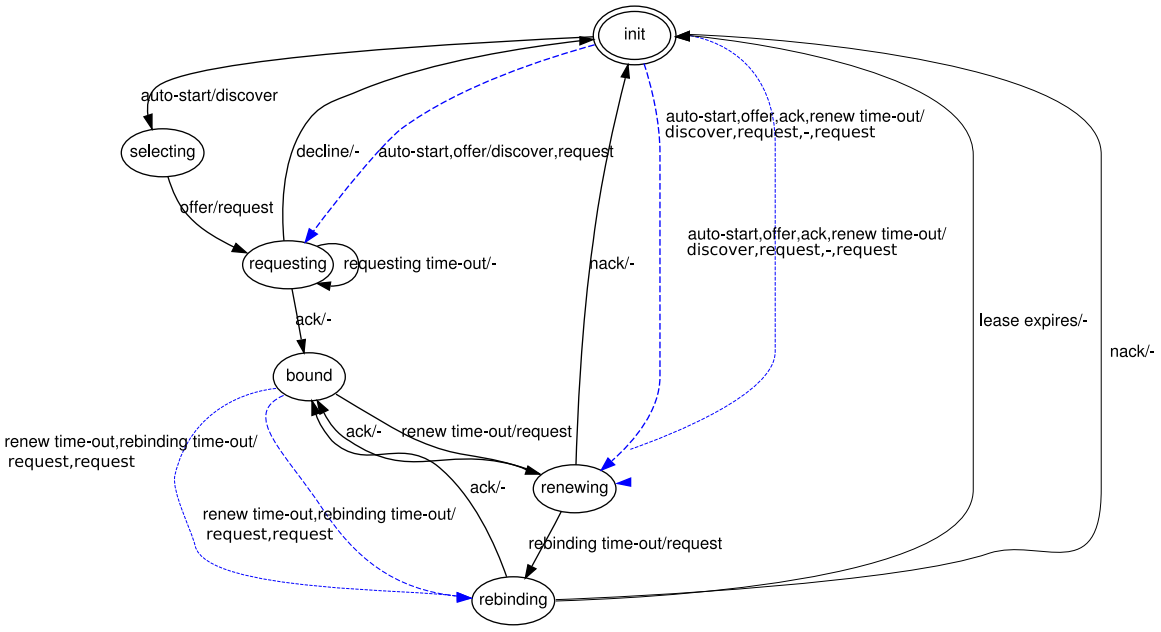


Figura 2.6: Especificação do protocolo DHCP com transições duplicadas

Algoritmo 2 Simetrização de uma especificação

1. Etiquete cada estado da especificação com um inteiro que representa a diferença entre o número de transições que saem e que chegam neste estado, isto é, com o seu respectivo graus. Esse número pode ser positivo, negativo ou zero. Assim, por definição, toda transição possui tanto uma origem como um destino, o que implica que a soma de todas as etiquetas deva ser zero.

2. Selecione um estado s_k com uma etiqueta negativa e um vértice s_l com uma etiqueta positiva. Encontre o menor caminho de s_k a s_l que atravesse o menor número de transições na especificação original. Duplique as arestas que estiverem nesse caminho. Atualize as etiquetas de s_k e s_l. As etiquetas nos estados intermediários não mudam.

3. Se a especificação aumentada for simétrica, o algoritmo termina. O custo do aumento é o número total de transições que foi acrescentada. Se a especificação continuar assimétrica, retorne para o passo 2.

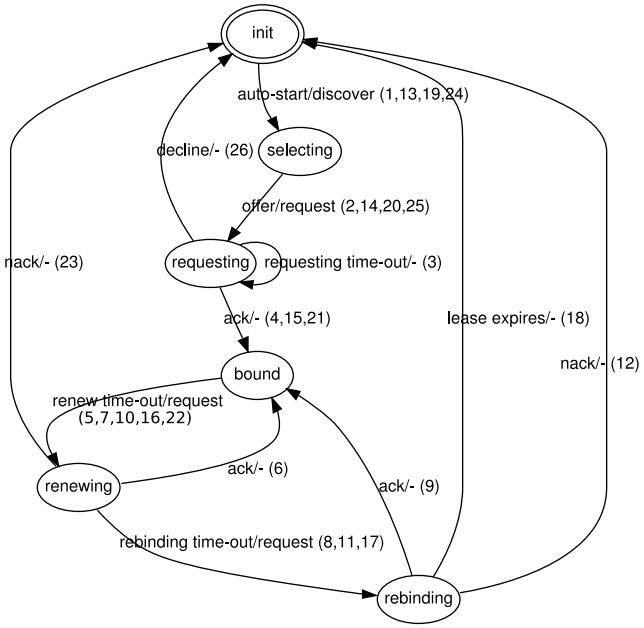


Figura 2.7: Passeio de transições do protocolo DHCP

A Figura 2.4 exibe a especificação do protocolo DHCP com o grau de cada um de seus vértices. Percebe-se que o protocolo não é simétrico. Logo, para gerar um passeio de transições faz-se necessário duplicar algumas de suas transições, conforme os passos do Algoritmo 2, cujo resultado é ilustrado na Figura 2.6. Uma vez feito isso pode-se então executar os passos do

Algoritmo 1. A arborescência geradora é mostrada na Figura 2.5 e o passeio de transições na Figura 2.7. A numeração ao lado das transições simboliza a ordem em que serão executadas nas seqüências de testes. Neste caso particular, a seqüência seria: ⟨ *auto-start/discover, offer/request, requesting time-out/-, . . . , offer/request, decline/-* ⟩.

O algoritmo de passeio de transições verifica se todas as transições de uma especificação podem ser exercitadas em uma dada IST por meio da análise dos sinais de saída que foram produzidos após o estímulo dos sinais de entrada. Logo, ao se executar em uma implementação uma seqüência de testes gerada pelo método T, estamos testando todas as possíveis falhas de saída existentes, que é único tipo de cobertura de falhas oferecido por este método.

Os métodos D, W e U, que serão vistos nas seções seguintes, utilizam técnicas de identificação de estados para melhorar a capacidade de detecção de falhas no teste de conformidade. Esses métodos procuram por seqüências de sinais que sirvam como assinaturas para os estados, que podem ser utilizadas para garantir que o estado alcançado por uma determinada seqüência de teste é realmente o estado desejado.

2.5.2 Cobertura de falhas de transição de estado

As técnicas de geração de seqüências de teste que cobrem falhas de transição de estado visam testar se o estado alcançado após a aplicação de uma dada seqüência de teste é igual ao estado esperado. Eventualmente podem existir vários caminhos para se chegar a um determinado estado. A técnica utilizada para encontrar, a partir do estado inicial, todos os possíveis caminhos para todos os estados é chamada de análise de alcançabilidade.

A análise de alcançabilidade é um processo de exploração de estados que se inicia a partir do estado inicial e recursivamente explora todas as transições. O resultado é o grafo de alcançabilidade. O grafo de alcançabilidade é uma árvore cuja raiz é o estado inicial. Essa árvore representa as diferentes possibilidades de se chegar a todos os estados de uma máquina de estados finita e determinista sem passar por transições repetidas. Os nós filhos de um determinado nó da árvore são os estados que são alcançáveis a partir desse estado. Existe uma aresta da árvore para cada transição da máquina de estado e toda aresta é etiquetada com o respectivo sinal de entrada que estimula a transição que ela representa. Um estado será representado por mais de um nó se existir mais de uma transição que parte do estado inicial e alcança aquele estado. De uma maneira resumida, a análise de alcançabilidade constitui-se dos seguintes passos:

1. Inicia-se uma fila de estados que contem inicialmente somente o estado inicial;
2. Enquanto a fila de estados não estiver vazia;
    - Retira-se o próximo estado da fila;

- Consideram-se todas as possíveis entradas que perturbam a máquina nesse estado;
- Em função das respostas da máquina à perturbação, identificam-se todos os estados alcançáveis a partir desse estado e adiciona-os á fila, exceto aqueles que já foram previamente explorados;

Uma busca em largura pode ser utilizada para representar os passos acima, como está ilustrado no Algoritmo 3. Esse algoritmo aplicado à máquina de estados especificada na Figura 2.8 gera o grafo de alcançabilidade ilustrado na Figura 2.9. Essa máquina de estados será utilizada no restante deste capítulo para explicar os algoritmos de cobertura de falha de transição de estado. Embora didática, essa máquina de estados não especifica um protocolo real. Por essa razão, sempre que possível serão utilizados nesta dissertação especificação de protocolos reais na explicação desse e dos próximos algoritmos. No caso específico de análise de alcançabilidade, o protocolo escolhido para aplicá-la é o protocolo TCP, cuja especificação encontra-se ilustrada na Figura 2.10 e cujo grafo de alcançabilidade encontra-se ilustrado na Figura 2.11.

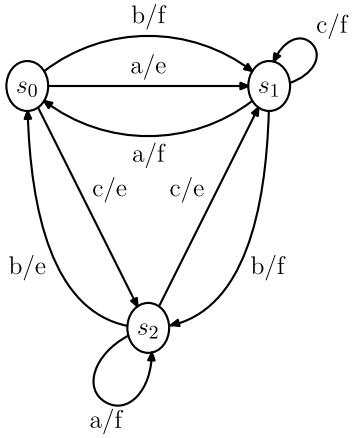


Figura 2.8: Especificação didática de exemplo

O que se obtém a partir do grafo de alcançabilidade são preâmbulos para todos os estados da máquina de estados em consideração. Um preâmbulo para um dado estado é uma seqüência de sinais de entrada e de saída que, quando aplicada a partir do estado inicial, conduzirá a máquina de estados a esse estado.

A análise de alcançabilidade tem sido aplicada também a teste de conformidade em contexto. Nesse caso, a máquina de estados sob a qual se aplica a análise é a resultante da composição de máquina de estados do componente e do contexto, que será discutida no próximo capítulo.

Algoritmo 3 Geração de grafo de alcançabilidade

Entrada: FSM
Saída: grafo de alcançabilidade

```
1: classe noGrafo:
2:     defina noGrafo (estado) :
3:         este.s = estado /* estado */
4:         este.rs = [ ] /* estados alcançáveis */
5:
6: incluso = { } /* dicionário */
7: para todo(a) s ∈ S faça
8:    incluso[s] = Falso
9: fim para
10: raiz = s_0
11: incluso[raiz] = Verdadeiro
12: grafo = novo noGrafo(raiz)
13: fila = [ ]
14: fila.anexa(grafo)
15: enquanto fila ≠ [ ] faça
16:    g = fila.desenfileira( )
17:    para todo(a) entrada, saída, ns ∈ adjacentes[g.s] faça
18:       se incluso[ns] então
19:          g.rs.anexa((ns, entrada))
20:       senão
21:          w = novo noGrafo(ns)
22:          g.rs.anexa ((w, entrada))
23:          fila.enfileira(w)
24:          incluso[ns] = Verdadeiro
25:       fim se
26:    fim para
27: fim enquanto
28: retorna grafo
```

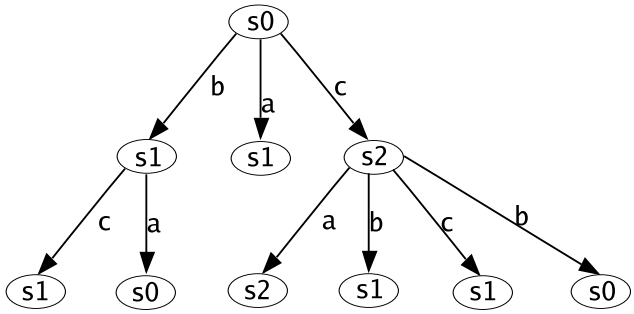


Figura 2.9: Grafo de alcançabilidade da especificação didática de exemplo

Uma vez que se tem preâmbulos para todos os estados de uma máquina de estados, ou seja, sabe-se como obter seqüências de sinais de entrada e saída para alcançar todos os estados a partir do estado inicial, pode-se aplicar uma técnica de cobertura de falhas de transição de estado. As três principais técnicas são: o método da seqüência de distinção de estados (método D ou DS) [Gonenc, 1970], que procura uma seqüência de sinais de entrada que produza uma seqüência de sinais de saída distinta para cada um dos estados; o método do conjunto de caracterização de estados (método W ou W-set) [Chow, 1978], que procura um conjunto de sinais de entrada que, quando aplicados individualmente cada um desses sinais a um dado estado da FSM, produzirá um conjunto de sinais de saída distinto dos demais estados; e o método de seqüências únicas de sinais de entrada e de saída (método U ou UIO) [Sabnani and Dahbura, 1988], que procura, para cada estado da FSM, uma seqüência única de sinais de entrada e saída que o distinga dos demais estados.

Tabela 2.1: Principais técnicas de cobertura de falhas de transição de estado

Método	**Estratégia**	$\mathbf{s_0}$	$\mathbf{s_1}$	$\mathbf{s_2}$
DS	Procurar uma seqüência de sinais de entrada que produza uma seqüência de sinais de saída distinta para cada um dos estados (e.g., $DS = \{a.a\}$).	{a/e.a/f}	{a/f.a/e}	{a/f.a/f}
W-set	Procurar um conjunto de sinais de entrada que, quando aplicados individualmente cada um desses sinais a um dado estado da FSM, produzirá um conjunto de sinais de saída distinto dos demais estados (e.g., *W-set* $= \{a, b\}$).	{a/e, b/f}	{a/f, b/f}	{a/f, b/e}
UIO	Procurar, para cada estado da FSM, uma seqüência única de sinais de entrada e saída que o distinga dos demais estados.	{a/e}	{c/f}	{b/e}

Aplicando-se esses três algoritmos à máquina de estados da Figura 2.8 obtém-se, para cada um dos estados dessa máquina, as assinaturas listadas na Tabela 2.1. A seqüência única

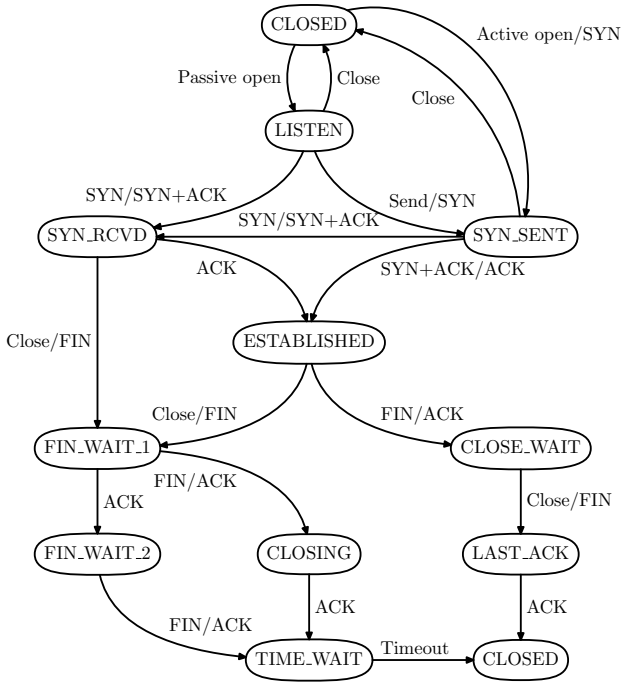


Figura 2.10: Especificação do protocolo TCP

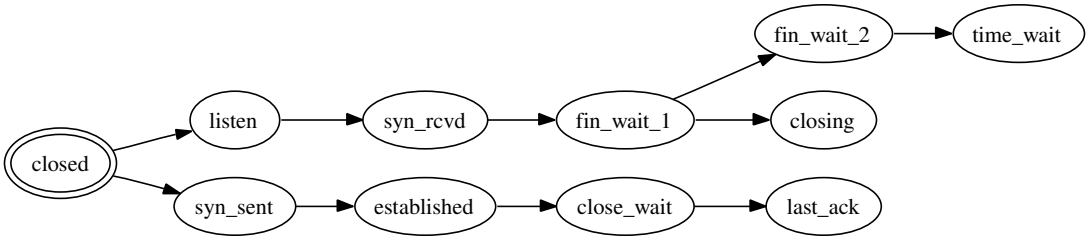


Figura 2.11: Grafo de alcançabilidade do protocolo TCP

de sinais de entrada {a.a} é uma seqüência de distinção para essa máquina, pois produz uma seqüência de sinais de saída diferente quando aplicada a cada um dos estados. O conjunto de sinais de entrada {a,b} é um conjunto de caracterização para essa máquina, pois quando aplicados individualmente cada um desses sinais a cada um dos estados, geram um conjunto de pares com um sinal de entrada e um sinal de saída distinto para cada um dos estados. Por último, as seqüências de sinais de entrada e saída {a/e}, {c/f} e {b/e} são, respectivamente, seqüências únicas de sinais de entrada e de saída para os estados s_0, s_1 e s_2, pois são geradas unicamente por cada um desses estados.

O protocolo especificado pela máquina de estados da Figura 2.8 é hipotético e foi utilizado apenas para fins ilustrativos. Na especificação de protocolos reais, dos três algoritmos descritos acima para teste de conformidade de protocolos, somente o método U e por vezes uma combinação do método W com o método U são viáveis. Por esta razão, será descrito com mais detalhes na próxima seção apenas o algoritmo para o método U.

2.5.3 Seqüências únicas de sinais de entrada e de saída (método U)

Uma seqüência única de sinais de entrada e de saída serve para determinar se uma certa IST encontra-se em um determinado estado. Nem sempre existem tais seqüências para todos os estados de uma dada especificação. Levando-se em consideração a existência dessas assinaturas, o melhor método para encontrá-las é enumerar todas as seqüências não repetidas de entradas e saídas e verificá-las com respeito à propriedade de seqüência única de entradas e saídas. O Algoritmo 4 consegue isso construindo uma árvore expandida exponencial de seqüências de entradas e saídas [Holzmann, 1990]. Os nós na árvore que estiverem a uma distância n da raiz correspondem a seqüências de entradas e saídas de tamanho n. Cada nó tem associado a si dois estados. O primeiro conjunto, P, contém um particionamento do conjunto dos estados de S em classes. Dois estados estão numa mesma classe do particionamento P se e somente se eles não puderem ser distinguidos um do outro através da aplicação da seqüência de entradas e saídas associada pelo nó: a especificação produz as mesmas saídas quando aplicadas essas seqüências, não importando quais desses estados são escolhidos como estado inicial. Os membros do segundo conjunto T definem para cada estado em S qual será o estado final se a seqüência de entradas e saídas representada pelo nó for aplicada àquele estado considerado como um estado inicial.

Seja $\delta(i, j)$ o estado que a IST deva chegar caso a entrada j seja recebida enquanto no estado i, e definamos $\lambda(i, j)$ o sinal de saída que é gerado durante a transição, se existir algum. Além disso, definamos $T[i]$ como o $i - ésimo$ elemento do conjunto T para o nó corrente e $T_j[i]$ o $i - ésimo$ elemento do conjunto T para seu $j - ésimo$ nó sucessor.

Algoritmo 4 Derivação de seqüências únicas de entradas e saídas

1. Inicialmente o particionamento P consiste de um simples conjunto que inclui todos os estados S da especificação. O conjunto T possui todos os membros. O valor inicial para o $i - ésimo$ membro de T, com $1 \leq i \leq S$, é i. A árvore de seqüências de entradas e saídas é iniciada com um simples nó, chamado o nó raiz da árvore, que corresponde à seqüência nula. (Um nó folha é um nó sem sucessores.)

2. Ordena-se os nós folhas da árvore em uma lista e remove-se os duplicados. Agora para cada nó folha associa-se V nós sucessores na árvore, um para cada sinal de entrada possível. Atribue-se $T_j[i] = \delta(T[i], j)$.

3. O particionamento P_j para o $j - ésimo$ nó sucessor é derivado do particionamento atual P da seguinte maneira. Define-se o conjunto O os sinais de saída associado com cada uma dos S estados da última transição. Considera-se cada classe em P separadamente. Produza uma lista de todos os sinais de saída distintos que são gerados pelos estados que estão inclusos na classe considerada. Esta classe em P é então quebrada em subclasses de tal forma que todos os estados que geraram o mesmo sinal de saída são associados à mesma subclasse. Se todos os estados dentro da classe considerada gerarem o mesmo sinal de saída para o último sinal de entrada aplicado, todos permanecem na mesma classe de particionamento.

4. Se há uma classe no particionamento de P neste ponto que contenha somente um estado, o nó que possui este particionamento definirá uma seqüência única de entradas e saídas para esse estado.

5. Os passos 2 a 4 são repetidos até que sejam encontradas seqüências únicas de entradas e saídas para todos os estados.

Este algoritmo procura seqüências únicas de entradas e saídas para todos os estados da especificação ao mesmo tempo, com as menores seqüências sendo inspecionadas primeiro. Em máquina de estados de protocolos reais é difícil encontrar seqüências únicas de entrada e saídas para todos os seus estados, como é o caso do protocolo TCP, cuja FSM encontra-se ilustrada na Figura 2.10 e cujas seqüências únicas de entradas e saídas estão descrita na Tabela 2.2.

Tabela 2.2: Seqüências únicas de entradas e saídas para o TCP

Estado	Seqüência única de entradas e saídas
CLOSED	Active open/SYN
LISTEN	Send/SYN
SYN_RCVD	ACK ; Close/FIN
SYN_SENT	SYN+ACK/ACK
ESTABLISHED	FIN/ACK ; Close/FIN
FIN_WAIT_1	FIN/ACK ; ACK
FIN_WAIT_2	*Não possui*
CLOSING	*Não possui*
TIME_WAIT	*Não possui*
CLOSE_WAIT	Close/FIN ; ACK ; Active open
LAST_ACK	*Não possui*

2.6 Notas bibliográficas

No modelo de FSM há várias questões sobre testes que já foram levantadas. Os problemas fundamentais de testes de protocolos são a seqüência de sincronização, a verificação de estado, a identificação de estados, a verificação da máquina ou teste de conformidade, e a identificação da máquina [Lee and Yannakakis, 1996].

Na seqüência de sincronização o objetivo é detectar em qual estado se encontra a máquina após a execução do teste. O problema já foi completamente solucionado por [Kohavi, 1990], [Moore, 1956], e [Eppstein, 1990].

Para o problema de identificação de estados, a questão é encontrar qual o estado corrente de uma determinada FSM foi alcançado. Este problema resolve-se utilizando uma seqüência de distinção para distinguir um estado dos demais; isso foi estudado por [Kohavi, 1990, Gill, 1962, Gill, 1961].

Outro problema correlacionado é o da verificação de estado, que consiste em verificar se uma dada FSM encontra-se em um determinado estado. O uso de uma seqüência única de entradas e saídas que funciona como uma assinatura para o estado em consideração é a solução

que melhor resolve este problema. Seqüências únicas de entradas e saídas foram estudadas por [Gill, 1961, Hsieh, 1971, Sabnani and Dahbura, 1988, Sidhu and Leung, 1989].

O problema de teste de conformidade também é correlacionado ao problema de verificação de estado, sendo que o objetivo deste é verificar todos os estados de uma máquina. O uso de passeios nas transições foi estudado em [Friedman and Menon, 1971] [Tsunoyama and Naito, 1981][Agrawal and Seth, 1988][Kohavi, 1990] e se desdobra no problema do caixeiro viajante [Garey and Johnson, 1979]. Uma solução que garante uma maior cobertura de falhas utiliza seqüências únicas de entrada e saída. A utilização de seqüências únicas de entrada e saídas para gerar seqüências de teste nem sempre é adequada, devido a geração de casos de testes de tamanhos intratáveis. Uma solução mais simples é a geração de um passeio que cubra as mesmas falhas que seriam cobertas pelas seqüências únicas de entrada e saída. Este problema é conhecido como o problema do caixeiro viajante rural [Aho and Lee, 1987] e é considerado um problema NP-difícil, o que significa que é pouco provável que haja uma algoritmo em tempo polinomial para resolvê-lo [Cormen et al., 1990].

Afora estes problemas, há outros relacionados a teste de protocolos cujo objetivo não é a geração de seqüências de teste. Nesse caso é feito um teste direto do protocolo, seja utilizando árvores de decisão, conhecido como testes adaptativos [Lee and Yannakakis, 1996], seja utilizando buscas em profundidade, como em [Fernandez et al., 1992, Holzmann, 1985].

Um estudo mais detalhado sobre o estado da arte em engenharia de protocolos pode ser encontrado em [Lee and Yannakakis, 1996] [Bochmann and Petrenko, 1994] [Sidhu and Leung, 1989] [Dssouli et al., 1999] [Lai, 2002]. A terminologia completa para o domínio de teste de protocolos pode ser encontrada na norma ISO9646 [Consortium, 1994].

Capítulo 3

Teste de Conformidade em Contexto

Este capítulo discute os principais algoritmos e técnicas disponíveis na literatura para resolver o problema de teste de conformidade em contexto. A primeira técnica vista é a composição de máquinas de estados, que é utilizado para descrever o comportamento combinado do componente e do contexto. Em seguida, uma família de algoritmos que utiliza técnicas de separação e avaliação seletiva é apresentada. Por último é apresentado um algoritmo de verificação direta.

3.1 Composição de máquinas de estados

Um modo de resolver o problema de teste de conformidade em contexto é aplicar os algoritmos descritos no capítulo anterior à máquina de estados que representa o comportamento conjunto do contexto e do componente. O Algoritmo 5 descreve os passos da composição de duas FSMs. Aplicando-o às especificações das Figuras 3.1(a) e 3.1(b) obtém-se a máquina de estados ilustrada na Figura 3.1(c).

O estado inicial da máquina composta é o estado em que o componente e o contexto encontram-se em seus respectivos estados iniciais. O número de estados na máquina composta é igual a $|S_T| \times |S_P|$ e o de número de transições é igual a $(|S_T| \times |\text{Transições}_P|) + (|S_P| \times |\text{Transições}_T|)$.

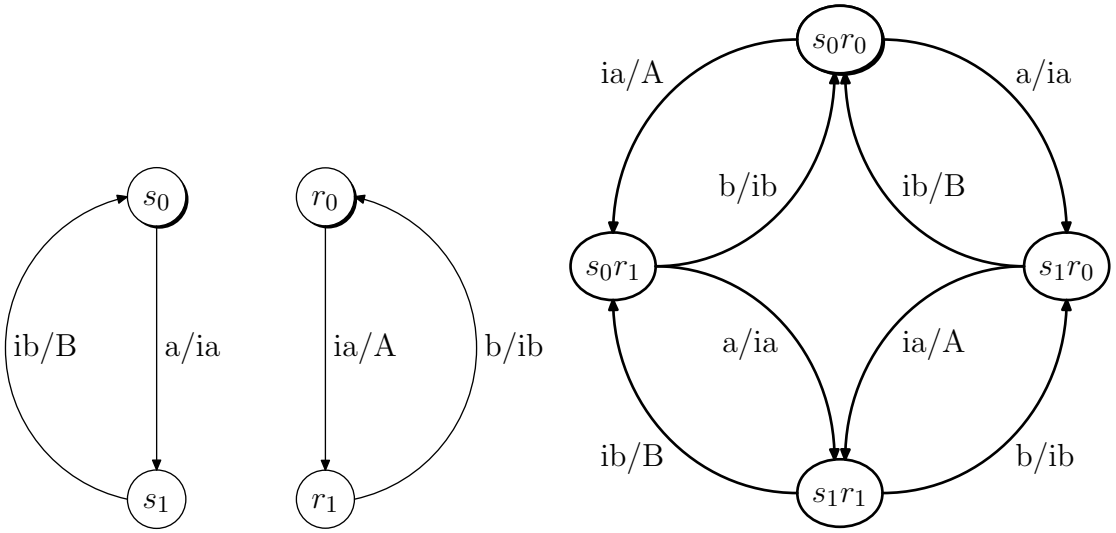
A geração do produto cartesiano de duas FSMs foi inicialmente descrita em
[Kohavi, 1990], no âmbito de circuitos digitais, e tem sido utilizada na literatura para teste em contexto
[Yevtushenko et al., 1998, Anido et al., 2003, Boroday et al., 2002].

Os estados globais obtidos a partir da composição podem ser classificados em: estáveis,

Algoritmo 5 Composição de FSMs

Entrada: T, P
Saída: $\Pi = T \times P$

1: /* Gera estados de Π */
2: **para todo(a)** $s_x \in S_T$ **faça**
3: **para todo(a)** $s_y \in S_P$ **faça**
4: Crie um estado $s_{(x,y)}$
5: Adicione $s_{(x,y)}$ a S_Π
6: **fim para**
7: **fim para**
8: /* Gera transições de Π */
9: **para todo(a)** $s_x \in S_T$ **faça**
10: **para todo(a)** transição (s_q, i, o) de s_x **faça**
11: **para todo(a)** $s_y \in S_P$ **faça**
12: Adicione uma transação $(s_{(q,y)}, i, o)$ a $s_{(x,y)}$
13: **fim para**
14: **fim para**
15: **fim para**
16: **para todo(a)** $s_y \in S_P$ **faça**
17: **para todo(a)** transição (s_r, i, o) de s_y **faça**
18: **para todo(a)** $s_x \in S_T$ **faça**
19: Adicione uma transação $(s_{(x,r)}, i, o)$ a $s_{(x,y)}$
20: **fim para**
21: **fim para**
22: **fim para**



(a) Contexto (b) Componente (c) Composição

Figura 3.1: Exemplo de composição de FSMs

instáveis, alcançáveis e inalcançáveis. Esta classificação é utilizada em técnicas de redução da máquina composta [Yevtushenko et al., 1998] e é apresentada nas definições seguintes:

Definições complementares

Definição 5 *Um estado global é um estado de* Π*, onde* Π *denota a máquina de estados resultante da composição das máquinas de estado do contexto e do componente. O conjunto de todos os estados globais é denotado por* $\Gamma = \{\sigma | \sigma \text{ é um estado de } \Pi\}$.

Definição 6 *Um estado global alcançável é um estado global que é alcançado durante a execução conjunta de* T *e* P*. O conjunto de todos os estados globais alcançáveis é denotado por* $\Re = \{\rho | (\rho \in \Pi,\ \rho \text{ é alcançado durante a execução conjunta de } T \text{ e } P)\}$.

Definição 7 *Um estado global não alcançável é um estado global que nunca acontecerá durante a execução conjunta das duas máquinas.*

Definição 8 *Um estado global estável é um estado global que* Π *chega após enviar uma resposta para o ambiente e antes de receber outro sinal do ambiente. O conjunto de todos os estados globais estáveis será denotado por*
$\Sigma = \{\tau | (\tau \in \Re,\ \tau \text{ é alcançado logo após o envio de um sinal para o ambiente})\}$.

A relação entre Γ, $\Re$ e Σ é ilustrada na Figura 3.2. O conjunto de estados transientes é ilustrado por $\Re - \Sigma$ (área hachurada da figura).

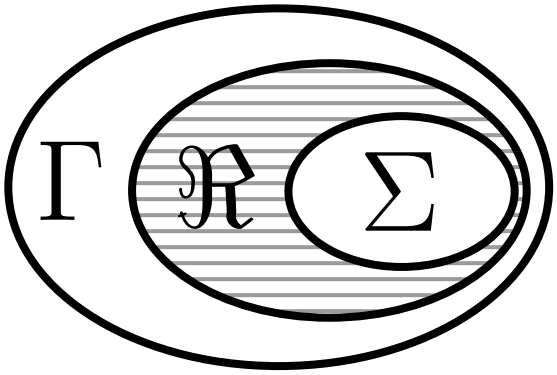


Figura 3.2: Relação entre conjuntos de estados

3.2 Teste por separação e avaliação progressiva

Esta seção descreve uma categoria de algoritmos cujo princípio é fazer passeios aleatórios guiados na máquina de estados do componente. O primeiro algoritmo foi publicado em [Lee et al., 1996] e foi chamado de guiado porque tentava passar pelas transições ainda não visitadas e foi chamado de aleatório porque, a priori, não utiliza nenhuma heurística para decidir quais transições de um estado é melhor exercitar. Em [Cavalli et al., 1999] é apresentado o algoritmo *hit-or-jump* como a generalização de três algoritmos básicos:

1. O algoritmo estruturado, no qual as seqüências de teste são geradas a partir da máquina de estados que representa o produto cartesiano das máquinas de estados do contexto e do componente;

2. O passeio aleatório[1], que utiliza um algoritmo guloso[2], na qual o próximo passo da seqüência de teste é decidido a partir do estado global atual, escolhendo-se aleatoriamente alguma transição ainda não exercitada; e

3. O passeio aleatório guiado, que segue a mesma idéia do algoritmo anterior, mas favorece nas escolhas as transições do componente embutido que está sob teste.

O algoritmo baseia-se em *hits* (acerto) ou *jumps* (saltos, no caso de não acerto). Um acerto é quando se alcança alguma transição ainda não exercitada do componente embutido. Caso contrário salta-se para outro nó da árvore de busca. As idéias apresentadas no algoritmo lembram as técnicas de avaliação e seleção (*branch and bound*) de pesquisa operacional. Nenhum modelo de cobertura de falhas é apresentado pelos seus autores.

[1] do Inglês *random walk*.

[2] Um algoritmo guloso é aquele que sempre toma a decisão que parece melhor no momento[Cormen et al., 1990].

Algoritmo 6 Separação e avaliação seletiva

Entrada: T e P

Saída: casos de testes

1: $r_{(k)} \leftarrow r_0$

2: $s_{(k)} \leftarrow s_0$

3: **enquanto** Houver transação não marcada de P **faça**

4:

1. **Acerto** - A partir do nó atual ($s_{(k)}$) conduz-se uma busca em $T \times P$ até que aconteça (a) ou (b):

   (a) Alcançou-se uma aresta que está associada com transições não marcadas na máquina componente P (um acerto). Então:

      i. Insere-se o caminho do nó atual para a aresta (inclusive) na seqüência de teste em construção;

      ii. Marca-se a nova transição exercitada de P;

      iii. Vai-se ao próximo nó ($s_{(k+1)}$);

      iv. Apaga-se o grafo alcançado;

      v. Volta-se ao passo 1.

   (b) Chega-se a uma profundidade ou um espaço limite sem encontrar nenhuma transição não marcada de P. Então move-se para 2.

2. **Salto** -

   (a) Constrói-se uma árvore de busca, cuja raiz é ($s_{(k)}$).

   (b) Examina-se todos os nós folhas da árvore, e seleciona um aleatoriamente;

   (c) Insere-se o caminho a partir da raiz até o nó folha na seqüência de teste;

   (d) Chega-se ao nó folha selecionado ($s_{(k+1)}$);

   (e) Volta-se ao passo 1.

5: **fim enquanto**

O Algoritmo 6 traça os passos do *hit-or-jump*. É requerido que o contexto T e o componente P estejam em seu respectivo estado inicial.

3.3 Teste direto

Os algoritmos apresentados nas seções anteriores tentam solucionar o problema de teste de conformidade garantindo o máximo, senão a completa, cobertura de falhas de transição e de sinais de saída. Uma heurística apresentada em [Fernandez et al., 1992] faz um balanceamento entre a porcentagem de possíveis falhas cobertas e a complexidade do procedimento de teste. A idéia é que, para uma grande classe de propriedades, armazenar todos os grafos de alcançabilidade não é obrigatório. É suficiente visitar todos os estados. Uma busca em profundidade do grafo de alcançabilidade do contexto realiza tal operação.

Esse método, conhecido como teste direto é um método intermediário, pois oferece um bom compromisso entre requisitos de espaço e tempo. As idéias apresentadas foram inicialmente propostas em [Holzmann, 1985] no contexto de verificação parcial como um possível método para restringir o espaço de estados durante uma busca dispersa e foram redescobertas em [Fernandez et al., 1992, Jéron, 1991] aplicadas para completar a verificação através do exame de modelos em linha. A idéia principal do método é atravessar uma espécie de produto síncrono de sistemas de transições finitas. O Algoritmo 7 descreve os passos da busca em profundidade utilizada para tal produto.

3.4 Notas bibliográficas

O problema de teste de protocolos em contexto foi inicialmente postulado por [Lee et al., 1993], e é também conhecido como teste de FSMs comunicantes. Antes do estudo de teste em contexto no domínio de engenharia de protocolos, o problema havia sido postulado para o domínio de circuitos digitais [Das and Farmer, 1975].

Em [Petrenko et al., 1996] o problema é novamente levantado e algumas peculiaridades são delineadas. Um algoritmo guloso é sugerido. Uma das deficiências apresentadas nesse trabalho é a falta de distinção entre mensagens internas e mensagens externas. Outro problema encontrado neste trabalho é explosão de estados.

Desde então vários trabalhos que tratam do problema de teste em contexto foram apresentados, para diferentes arquiteturas e configurações. As configurações mais consideradas na literatura são relativas à expansão ou ao determinismo das Máquinas de Estados Finitos com sinais de entrada e saída (FSMs). Expansão está relacionada à parametrização dos sinais de

Algoritmo 7 Verificação direta

Entrada: FSM
Saída: casos de teste

```
1: visitado = ∅
2: para todo(a) s ∈ S faça
3:     visitado[s] = Falso
4: fim para
5: raiz = s_0
6: visitado[raiz] = Verdadeiro
7: pilha = [ ]
8: pilha.anexa(raiz)
9: enquanto pilha ≠ [ ] faça
10:    s = pilha.desempilha( )
11:    para todo(a) ns ∈ adjacentes[s] faça
12:        se não visitado[ns] então
13:            visitado[ns] = Verdadeiro
14:            pilha.anexa(ns)
15:        fim se
16:    fim para
17: fim enquanto
```

entrada e saída. Essa expansão pode induzir ao indeterminismo de uma FSM, que acontece quando pelo menos um estado da FSM possuir duas transições com o mesmo sinal de entrada. Se, para estas transições, houver um sinal diferente de saída, a FSM é dita não determinista e observável [Boroday et al., 2002].

Após o trabalho de [Petrenko et al., 1996] surgiu o trabalho de [Lee et al., 1996], onde é apresentada uma heurística que evita o problema de explosão de estados, através de um passeio aleatório, cujo nome é *guided random walk*. Nesse trabalho já pode ser observado o tratamento de transições internas.

Em seguida surgiu o trabalho de [Yevtushenko et al., 1998], onde o problema de explosão de estados não é tratado e nem mencionado. Baseados nas transições internas, os autores mostram uma heurística para a minimização da FSM composta, que consiste na remoção das transições que possuam somente sinais internos. A idéia de remoção de transições internas foi inicialmente proposta em [Holzmann, 1990].

O trabalho apresentado em [Yevtushenko et al., 1998] considera somente FSMs deterministas. Um algoritmo de minimização das seqüências de teste da FSM composta para o caso de composição de FSMs não deterministas, é apresentado em [Anido et al., 2003]. A descrição de outras técnicas de redução da máquina composta pode ser encontrada em

[do Rocio Senger de Souza et al., 2001].

Retornando ao problema de explosão de estados, um novo trabalho [Cavalli et al., 1999] é apresentado com os mesmo conceitos de *guided random walk*[Lee et al., 1996]. Os autores chamam o algoritmo de *hit-or-jump*, cuja idéia é desviar enquanto não encontrar uma transição do contexto que estimule uma transição no componente, dado um limite de profundidade predeterminado. A deficiência desse trabalho é a não diferenciação entre o *hit-or-jump* e o *guided random walk*, exceto o limite de profundidade da busca. Outra deficiência do trabalho é a forte ligação entre o algoritmo e a ferramenta ObjectGEODE[Verilog, 2002], onde os *hits* (acertos) devem ser definidos manualmente pelo testador, mediante o uso de condições de paradas, quebrando o conceito de automatização da geração das seqüências de teste.

Ainda tratando de composição de FSMs, em [Boroday et al., 2002] é estudada a combinação de cobertura de falhas e cobertura da especificação. Os operadores de mutação, descritos em [Fabbri and Maldonado, 1994], são utilizados. A idéia principal do trabalho é procurar uma "seqüência de configuração" que distinga o sistema de FSMs comunicantes dos sistemas de FSMs comunicantes mutantes. A "seqüência de configuração" consiste no mesmo conceito de seqüência única de entrada e saída [Gill, 1961, Hsieh, 1971, Sabnani and Dahbura, 1988, Sidhu and Leung, 1989], aplicado ao sistema de FSMs, restrita a um tamanho máximo. A idéia de minimização de FSM composta apresentada em [Yevtushenko et al., 1998, Anido et al., 2003, Holzmann, 1990] é sugerida.

Todos os trabalhos apresentados nas referências acima levam em consideração somente a parte de controle, ou os sinais das FSM. Para o caso de FSM estendidas, há a preocupação de se testar também a parte de dados. Em [Ural and Williams, 1993, Henniger and Ural, 2000] há um estudo sobre dependência de dados entre os estados, e em [Fecko et al., 2000] é feito um estudo de FSM temporizadas.

Considerando-se somente a parte de controle, os trabalhos existentes da área de teste em contexto ou teste de FSMs comunicantes podem ser classificados como: as abordagens explosivas, que são aquelas que fazem a composição de FSM, apresentando problema de explosão de estados [Petrenko et al., 1996, Yevtushenko et al., 1998, Holzmann, 1990, Anido et al., 2003, Boroday et al., 2002]; e as abordagens cegas ou aleatórias, que fazem passeios aleatórios nas especificações [Lee et al., 1996, Cavalli et al., 1999].

Capítulo 4

Avaliações Práticas

Neste capítulo é feita uma análise da aplicabilidade de teste em contexto para diferentes protocolos das redes TCP/IP(*Transmission Control Protocol / Internet Protocol*) [Postel, 1980]. O objetivo principal é avaliar a viabilidade de teste de conformidade em contexto no âmbito de engenharia de protocolos, onde métodos formais são bastante utilizados. Primeiramente será avaliada a pilha de protocolos do TCP/IP, considerando uma camada como um contexto e qualquer camada imediatamente subjacente como um componente. Posteriormente será feita uma avaliação do cliente e do servidor dos protocolos HTTP (Protocolo de Transferência de HiperTexto) e HTTPS (HTTP Seguro) via um servidor Proxy, onde este é o contexto e aqueles, respectivamente, são o ambiente e o componente. Por último, uma avaliação do cliente e servidor do protocolo DHCP via agente de entrega, tendo este como contexto.

4.1 Pilha de protocolos do TCP/IP

Conceitualmente a pilha de protocolos do TCP/IP não permite efetuar teste de conformidade em contexto. Na Figura 4.1 pode-se imaginar o teste do protocolo TCP a partir do protocolo HTTP (*HyperText Transfer Protocol*) [Fielding et al., 1999], ou o protocolo IP a partir do TCP ou Ethernet a partir do IP. Testar TCP a partir de HTTP seria o mesmo que testar um protocolo baseado em estados através de outro que não mantém estado. Além disso, mensagens do TCP não possuem mapeamento direto para mensagens HTTP. O mesmo raciocínio é válido para os demais pares de protocolos listados.

A razão é que os protocolos são geralmente descritos usando uma estrutura em camadas, onde cada camada implementa a comunicação a um nível diferente de abstração. Diferentes camadas manipulam tipos diferentes de dados e de formatos (de mensagens multi-

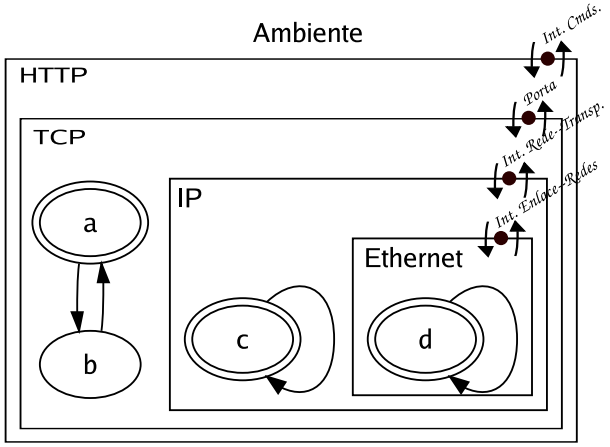


Figura 4.1: Modelagem hipotética do TCP/IP para teste em contexto

campos nas camadas superiores a bits nas camadas mais inferiores) e estão tipicamente sujeitas a diferentes restrições de tempo. A ISO definiu um padrão (Modelo de Referência OSI) que classifica as funções típicas de protocolos em sete camadas distintas [Kurose and Ross, 2002, da Silva Jr. et al., 2000]. Logo, não é possível mapear individualmente os protocolos das camadas do TCP/IP como máquinas de Mealy, descritas na Seção 2.2.

Portanto, o teste de conformidade em contexto não pode ser aplicado à pilha de protocolos do TCP/IP. Supõe-se que o teste de conformidade em contexto possa ser aplicado somente se forem considerados protocolos que trabalham em uma mesma camada e, por conseqüência, que trabalhe com o mesmo tipo de dados[1]. Esta afirmação é parcialmente verdadeira, pois não se consegue testar um cliente de um protocolo X por meio de um servidor de um outro protocolo Y. Por exemplo, não seria possível testar um servidor do protocolo HTTP por meio de um cliente FTP (*File Transfer Protocol*), ou o TCP a partir do UDP (*User Datagram Protocol*) e vice-versa. O próximo passo da análise, logo, é avaliar a aplicação de teste de conformidade em contexto em um protocolo de uma camada específica da pilha TCP/IP. Na próxima seção serão avaliadas a comunicação HTTP e HTTPS por meio de um servidor Proxy e a comunicação entre um cliente e um servidor DHCP intermediada por um agente de entrega.

[1]Esta afirmação é válida para outros sistemas de qualquer natureza, pois "os sistemas, freqüentemente, são hierárquicos, no sentido de que eles incluem outros sistemas. [...] [Um] mesmo sistema pode ser utilizado em diferentes sistemas. Seu comportamento em um sistema específico, contudo, depende de sua relação com outros sistemas."[Sommerville, 1993]

4.2 Comunicação HTTP com Servidor Proxy

O protocolo HTTP é atualmente um dos protocolos mais utilizados na Internet. A escolha do HTTP com Proxy para avaliação de teste de conformidade em contexto deve-se à sua simplicidade. Clientes do HTTP e do Proxy HTTP fazem requisições a seus respectivos servidores por meio do próprio protocolo HTTP.

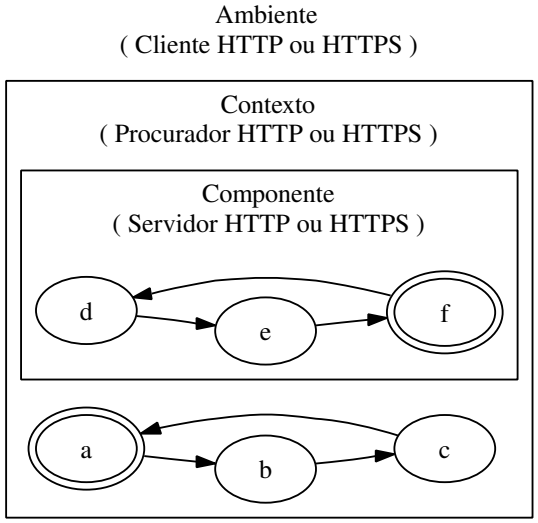


Figura 4.2: Modelagem do HTTP com Proxy para teste em contexto

Utilizando a terminologia da Seção 2.2 e tomando como base a Figura 4.2 teremos as entradas e saídas das entidades envolvidas no teste em contexto com Proxy HTTP descritas na Tabela 4.1. O ambiente considerado é um navegador Web, o contexto é um servidor Proxy e o componente é um servidor HTTP. O ambiente foi monitorado com o analisador de protocolos Ethereal [Hards, 2004].

Tabela 4.1: Entradas e saídas de HTTP com Proxy para teste em contexto

	Entrada (I_X)	**Saída** (O_X)
Ambiente (X=A)	• URL no campo endereço do navegador • Resposta HTTP	• Página Html requisitada • Requisição HTTP
Contexto (X=T)	• Requisição HTTP • Resposta HTTP	• Requisição HTTP • Resposta HTTP
Componente (X=P)	• Requisição HTTP • Resposta HTTP	• Requisição HTTP • Resposta HTTP

As requisições HTTP, de uma maneira em geral, são feitas pelo método GET, seguido do nome do objeto solicitado, do nome e da versão do protocolo. A Tabela 4.2 lista todos os métodos que podem ser utilizados em requisições HTTP [Tanenbaum, 2003]. A resposta é dada por um cabeçalho que contém o nome do protocolo, sua versão, o código de status da resposta e uma breve descrição do tipo de resposta. O código de status constitui-se de três dígitos que identificam se a solicitação foi satisfeita e, caso negativo, porque não. O primeiro dígito é utilizado para dividir a resposta em cinco grandes grupos, como mostrado na Tabela 4.3 [Tanenbaum, 2003]. O que vem após o cabeçalho, isto é, o conteúdo da mensagem transportada pelo HTTP, será irrelevante para a análise do protocolo.

Tabela 4.2: Métodos de Requisição do HTTP

Método	**Descrição**
GET	Solicita a leitura de uma página HTML
HEAD	Solicita a leitura do cabeçalho de uma página HTML
PUT	Solicita o armazenamento de uma página
POST	Anexa a um determinado recurso (e.g., uma página HTML)
DELETE	Remove a página HTML
TRACE	Ecoa a solicitação que foi enviada
CONNECT	Reservada para tunelamento de conexões SSL
OPTIONS	Consulta determinadas opções

Tabela 4.3: Grupos de Status de Resposta do HTTP

Código	**Significado**	**Exemplos**
1xx	Informação	100 = servidor concorda em manipular requisição do cliente
2xx	Sucesso	200 = solicitação foi um sucesso; 204 = nenhum conteúdo presente
3xx	Redirecionamento	301 = página movida; 304 = a página em cache ainda é válida
4xx	Erro do cliente	403 = página negada; 404 = página não encontrada
5xx	Erro do servidor	500 = erro interno no servidor; 503 = tente novamente mais tarde

Expostas as entradas e saídas do ambiente, do componente e do contexto, será feita uma análise formal da viabilidade de se aplicar teste em contexto em uma comunicação HTTP via Proxy. De acordo com a Tabela 4.1, observa-se que $O_A \cap I_T \cap I_P \neq \emptyset$, isto é, caso não existisse o contexto, seria possível fazer o componente interagir diretamente com o ambiente. Por raciocínio análogo, $I_A \cap O_T \cap O_P \neq \emptyset$. Em outras palavras, a requisição de um objeto via HTTP se propaga do cliente ao Proxy e do Proxy ao servidor alvo. A resposta do objeto solicitado se dá no sentido inverso. Devido ao protocolo HTTP ser sem estados, pode-se representar cada um dos entes envolvidos como uma máquina de um único estado. O controle e a observação do sistema foram feitos com um cliente HTTP (navegador) e com o analisador de protocolos

Ethereal [Hards, 2004]. A ausência de impasse, a completude, a atividade, a realização de progresso, a terminação, a correção parcial, a minimalidade e a estabilidade do protocolo HTTP já são conhecidas e validadas pela sua ampla utilização. Nessas condições, conclui-se que uma comunicação HTTP via um servidor Proxy é uma aplicação real e possível de efetuar teste de conformidade em contexto.

A validação das conclusões apresentadas foi feita em dois cenários distintos:
a) configuração explícita de um servidor Proxy; e b) utilização de um servidor Proxy transparente. Tanto neste quanto naquele o teste em contexto mostrou-se viável com o protocolo HTTP. De maneira similar, espera-se encontrar o mesmo resultado para Proxy com o protocolo HTTPS – o HTTP com uma camada de segurança. Esta questão será analisada na próxima seção.

4.3 Comunicação HTTPS com Servidor Proxy

A análise de teste em contexto em uma comunicação HTTPS via Proxy é um caso particularmente interessante por envolver dois protocolos distintos na camada de aplicação da pilha de protocolos do TCP/IP. O HTTPS é uma versão do HTTP que possui uma camada de segurança e é o que garante a segurança das transações de comércio eletrônico existentes na Internet. Por questão de brevidade, evitaremos entrar em detalhes sobre o protocolo. Suficiente dizer que o modelo adotado é similar ao da seção anterior: o cliente HTTPS é o ambiente onde faremos as requisições e receberemos respostas; o servidor Proxy é o contexto; e o servidor HTTPS o componente a ser testado. Devido às peculiaridades da configuração utilizada no teste, o analisador de protocolos foi executado no contexto. Para facilitar a análise dos protocolos, foram atribuídos nomes às máquinas envolvidas na comunicação: mafalda para o ambiente, susanita para o contexto e manolito para o componente.

Foram duas as configurações de Proxy utilizadas nos testes – as mesmas utilizadas na seção anterior. A primeira, uma configuração explícita do Proxy que, em outras palavras, é uma maneira de deixar o ambiente ciente que estava se comunicando com o sistema embarcado por meio de um contexto. A segunda, uma configuração em que o ambiente imaginava estar se comunicando diretamente com o componente embarcado quando, na verdade, a comunicação era interceptada por um contexto. Esta última configuração foi obtida por meio de Proxy transparente.

A configuração com Proxy explícito para HTTPS gerou como resultados a Tabela 4.4. Suficiente dizer que a comunicação se deu com sucesso. À primeira vista poderia se concluir que este seria mais um exemplo real de teste em contexto. Pelo contrário, uma comunicação HTTPS via servidor Proxy é um contra-exemplo para teste de conformidade em contexto, como será explicado no próximo exemplo.

Tabela 4.4: Interação Ambiente–Componente–Contexto com Proxy Explícito

No.	Tempo	Origem	Destino	Proto	Info
1	0.000000	mafalda	susanita	TCP	24936 > squid [SYN] Seq=0 Ack=0 Win=5840
2	1.999.560	susanita	mafalda	TCP	squid > 24936 [SYN ACK] Seq=0 Ack=1 Win=5792
3	1.999.521	mafalda	susanita	TCP	24936 > squid [SYN] Seq=0 Ack=0 Win=23360
4	1.999.607	susanita	mafalda	TCP	squid > 24936 [ACK] Seq=0 Ack=1 Win=23168
5	3.999.182	mafalda	susanita	TCP	24936 > squid [ACK] Seq=1 Ack=1 Win=5840
6	3.999.186	mafalda	susanita	TCP	[TCP Dup ACK 9#1] 24936 > squid [ACK] Seq=1 Ack=1 Win=5840
7	3.999.189	mafalda	susanita	HTTP	CONNECT manolito:443 HTTP/1.1
8	3.999.282	susanita	mafalda	TCP	squid > 24936 [ACK] Seq=1 Ack=178 Win=6864
9	3.999.728	susanita	manolito	TCP	1060 > https [SYN] Seq=0 Ack=0 Win=5840
10	3.999.920	manolito	susanita	TCP	https > 1060 [SYN ACK] Seq=0 Ack=1 Win=5792
11	3.999.959	susanita	manolito	TCP	1060 > https [ACK] Seq=1 Ack=1 Win=5840
12	4.000.112	susanita	mafalda	HTTP	HTTP/1.0 200 Connection established
13	5.998.851	mafalda	susanita	TCP	24936 > squid [ACK] Seq=178 Ack=40 Win=5840
14	5.998.854	mafalda	susanita	HTTP	Continuation or non-HTTP traffic
15	5.999.141	susanita	manolito	SSLv2	Client Hello
16	5.999.302	manolito	susanita	TCP	https > 1060 [ACK] Seq=1 Ack=106 Win=5792
17	6.029.214	manolito	susanita	TLS	Server Hello Certificate
18	6.029.245	susanita	manolito	TCP	1060 > https [ACK] Seq=106 Ack=1449 Win=8736
19	6.029.252	manolito	susanita	TLS	Server Key Exchange Server Hello Done
20	6.029.261	susanita	manolito	TCP	1060 > https [ACK] Seq=106 Ack=1487 Win=8736
21	6.029.522	susanita	mafalda	HTTP	Continuation or non-HTTP traffic
22	6.029.535	susanita	mafalda	HTTP	Continuation or non-HTTP traffic
23	7.998.515	mafalda	susanita	TCP	24936 > squid [ACK] Seq=283 Ack=1526 Win=8736
24	7.998.758	susanita	manolito	TLS	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
25	8.019.793	manolito	susanita	TLS	Change Cipher Spec, Encrypted Handshake Message
26	8.019.821	susanita	manolito	TCP	1060 > https [ACK] Seq=304 Ack=1546 Win=8736
27	9.998.427	susanita	manolito	TLS	Application Data
28	9.999.262	manolito	susanita	TLS	Application Data
29	9.999.292	susanita	manolito	TCP	1060 > https [ACK] Seq=821 Ack=1759 Win=8736
30	11.998.110	susanita	manolito	TLS	Application Data
31	11.999.923	manolito	susanita	TLS	Application Data
32	11.999.953	susanita	manolito	TCP	1060 > https [ACK] Seq=1178 Ack=2100 Win=8736

Os testes efetuados com Proxy transparente em uma comunicação HTTPS geraram como resultado a Tabela 4.5. Comparando-se a quantidade de pacotes exibidos nesta tabela com a quantidade de pacotes da Tabela 4.4, percebe-se uma diferença de vinte e sete pacotes. Do lado do cliente, a resposta observada foi que a página requerida não foi obtida. O resultado do teste é que não é possível este tipo de comunicação. O cliente chega a estabelecer uma conexão com o servidor Proxy (pacotes de número 1 ao 5 da Tabela 4.5), porém não obtém resposta à sinalização de abertura da conexão HTTPS (em contraste ao que acontece na Tabela 4.4, nos pacotes número 7, 12, 15, 24, 27 e 28). No caso de Proxy transparente com HTTPS os pacotes são interceptados pelo Proxy em nível de aplicação. No entanto, o servidor Proxy não emite nenhuma mensagem de erro e nem direciona o tráfego para o servidor HTTPS. Isto é evidenciado por não ter havido tráfego na comunicação entre o Proxy e o servidor HTTPS.

Existem razões de segurança para que isto ocorra. O protocolo HTTPS foi desenvolvido para evitar ataques do tipo *man-in-the-middle*. Uma análise formal deste problema criptográfico pode ser visto em [Menezes et al., 1996]. Sob a ótica de teste em contexto, temos um contexto permissivo, porém intrusivo. Logo, uma comunicação HTTPS interceptada por um servidor Proxy de maneira transparente é um contra-exemplo para teste de conformidade em contexto.

Tabela 4.5: Interação Ambiente–Componente–Contexto com Proxy Transparente

No.	**Tempo**	**Origem**	**Destino**	**Proto**	**Info**
1	0.000150	mafalda	manolito	TCP	11319 > https [SYN] Seq=0 Ack=0 Win=5840
2	0.720172	manolito	mafalda	TCP	https > 11319 [SYN ACK] Seq=0 Ack=1 Win=5792
3	2.719804	mafalda	manolito	TCP	11319 > https [ACK] Seq=1 Ack=1 Win=5840
4	2.719808	mafalda	manolito	SSLv2	Client Hello
5	2.719905	manolito	mafalda	TCP	https > 11319 [ACK] Seq=1 Ack=106 Win=5792

4.4 DHCP com Agente de Entrega

A escolha do protocolo DHCP para avaliação de teste em contexto deve-se ao fato de o protocolo ter ampla utilização e, principalmente, pelo protocolo manter informações de estado. Ele é bastante utilizado em redes TCP/IP cabeadas e sem fio para que computadores recém-conectados à rede possam ter acesso à Internet. Durante o processo de obtenção, renovação e liberação de um endereço IP um cliente DHCP pode passar por algo em torno de seis estados principais e 11 transições, dependendo da implementação. O servidor mantém estados para cada endereço IP da faixa que ele oferece, que podem ser: “em uso”, quando o endereço foi liberado a um cliente e ainda não teve seu tempo de liberação expirado, e “livre”, caso contrário. Even-

tualmente no meio da comunicação pode existir um ou mais agentes de entrega, que entregam pacotes entre o cliente e o servidor quando estes residem em redes físicas distintas.

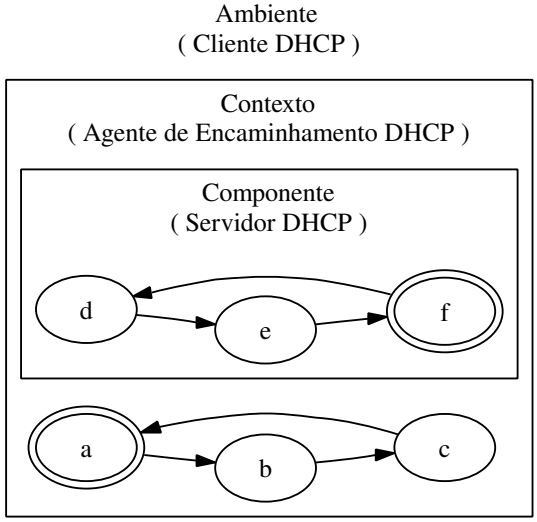


Figura 4.3: Modelagem do DHCP com *relay-agent* para teste em contexto

O cenário utilizado para a avaliação foi montado considerando-se um agente de entrega (*relay-agent*) que intermediava a comunicação cliente–servidor (Figura 4.3). Os equipamentos utilizados na avaliação estão ilustrados na Figura 4.4. Somente o servidor DHCP e a switch, que serviu também de roteador, tinham endereço IP fixo. O servidor ficou fisicamente separado de alguns clientes. Duas redes locais virtuais (VLAN) foram criadas para este propósito. As setas tracejadas com um endereço IP acima indicam o endereço obtido do servidor para cada cliente.

A máquina de estados (FSM) do cliente DHCP é mostrada na Figura 4.5. A descrição de seus estados encontra-se na Tabela 4.7. Os eventos que provocam transição de estado são as mensagens do protocolo, descritas na Tabela 4.6, e temporizadores (*time-outs*), que marcam o tempo máximo que um cliente aguardará uma mensagem de resposta do servidor.

A simulação dos eventos que geram a troca de mensagens entre o cliente e o servidor DHCP foi feita por meio de ferramentas presentes nas plataformas Linux e Windows. No Linux foi utilizado o comando *dhcpcd (DHCP client daemon)*. No Windows, *ipconfig*. Destes foram utilizados os argumentos listados na Tabela 4.8. Alguns eventos não são disparados pela simples manipulação desses comandos nos clientes. Para alguns casos foi necessário pré-configurar adequadamente o ambiente de teste para obter as mensagens desejadas. Um exemplo é simular que já existe alguma interface de rede com o mesmo número IP na rede. A captura do tráfego de pacotes DHCP também foi feita com o analisador de protocolos Ethereal, que foi executado

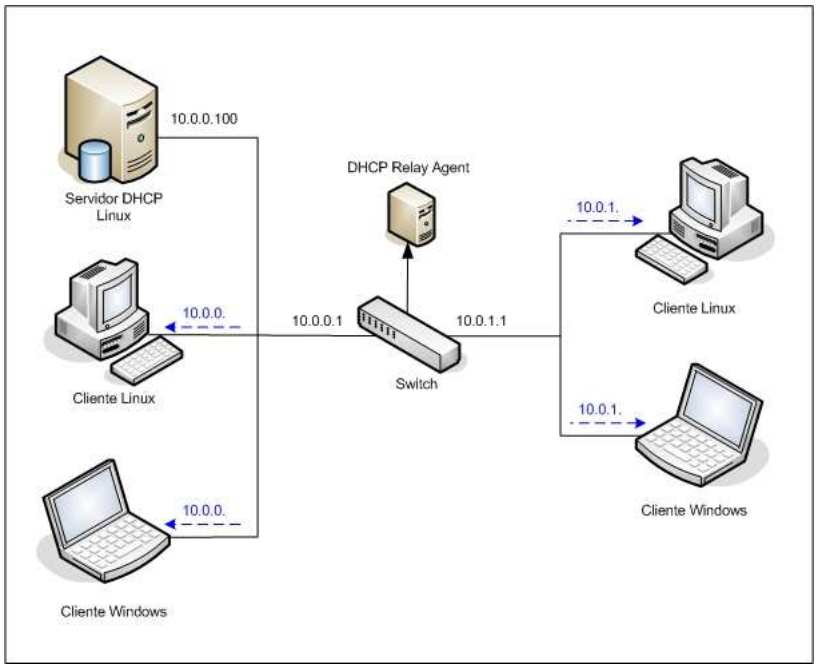


Figura 4.4: Distribuição dos computadores utilizados no teste

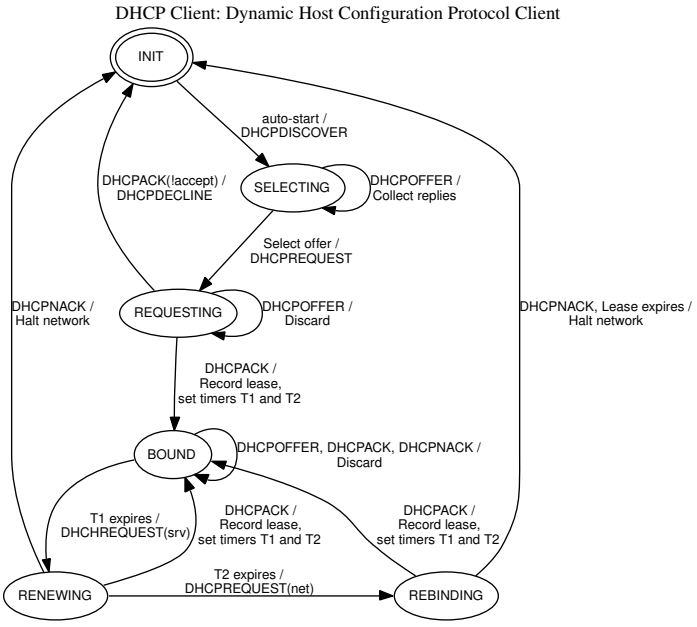


Figura 4.5: Máquina de estados do cliente DHCP

nas interfaces de rede dos computadores cliente e servidor.

Tabela 4.6: Descrição das mensagens do protocolo DHCP

Mensagem	**Uso**
DHPCDISCOVER	Cliente difunde uma mensagem para localizar servidores disponíveis.
DHCPOFFER	Servidor envia ao cliente em resposta a uma mensagem DHCPDISCOVER com oferta de parâmetros de configuração. .
DHCPREQUEST	Mensagem do cliente ao servidor para: (a) Solicitar parâmetros oferecidos de um servidor e implicitamente recusar a oferta de todos os outros; (b) Confirmar a validade de um endereço previamente alocado, por exemplo, após uma reinicialização; (c) Estender uma concessão para um endereço de rede particular.
DHCPACK	Mensagem do servidor para o cliente com parâmetros de configuração, incluindo o endereço de rede consignado.
DHCPNACK	Mensagem do servidor para o cliente indicando que o conhecimento que o cliente tem do endereço de rede está incorreto (por exemplo, o cliente pode ter se movido para uma outra subrede) ou o tempo de concessão expirou.
DHCPDECLINE	Mensagem do cliente ao servidor indicando que o endereço de rede já se encontra em uso.
DHCPRELEASE	Mensagem do cliente para o servidor renunciando o endereço de rede e cancelando o tempo de concessão remanescente.
DHCPINFORM	Mensagem do cliente ao servidor pedindo por configurações locais . O cliente já teve seu endereço IP configurado externamente.

A captura do tráfego DHCP no cliente e no servidor gerou os pacotes exibidos, respectivamente, nas Tabelas 4.9 e 4.10. A Figura 4.6 ilustra a troca de mensagens entre cliente, agente de entrega (*relay-agent*) e servidor. Percebe-se pelos resultados que é viável executar teste em contexto em uma comunicação entre cliente e servidor DHCP intermediada por um agente de entrega.

A explicação formal para esta viabilidade é a intersecção entre mensagens do agente, do cliente e do servidor. Em outras palavras, $O_A \cap I_T \cap O_T \cap I_P \neq \emptyset$ e $I_A \cap O_T \cap I_T \cap O_P \neq \emptyset$.

Tabela 4.7: Descrição dos estados do protocolo DHCP

Estado	**Descrição**
INIT	Estado inicial
SELECTING	Selecionando oferta de algum servidor DHCP
REQUESTING	Solicitando configurações da rede
BOUND	Endereço IP e outros parâmetros já obtidos pelo cliente
INITREBOOT	Estado inicial da máquina após sua inicialização
REBOOTING	Estado que indica que a máquina cliente foi reinicializada
RENEWING	Estado que indica renovação do endereço IP
REBINDING	Estado que indica re-associação de um endereço IP a um cliente

Tabela 4.8: Comandos para envio de mensagens DHCP

dhcpd	
-k	Envia o sinal SIGHUP ao processo dhcpcd, que envia a mensagem DHCPRELEASE ao servidor DHCP e destrói o cache de configurações de rede.
-n	Envia o sinal SIGLRM ao processo dhcpcd, que força a renovação do *lease time* emitindo uma mensagem DHCPREQUEST ao servidor.
-s	Envia uma mensagem DHCPINFORM ao servidor.
ipconfig	
/renew	Faz a requisição da concessão ou renovação de endereço IP.
/release	Faz a liberação da concessão de endereço IP.

Tabela 4.9: Captura na subrede do servidor

No.	Tempo	Origem	Destino	Proto	Info
1	0.000000	10.0.0.1	10.0.0.100	DHCP	DHCP Discover - Transaction ID 0xd334d534
2	0.016527	10.0.0.100	10.0.1.1	DHCP	DHCP Offer - Transaction ID 0xd334d534
3	0.020341	10.0.0.1	10.0.0.100	DHCP	DHCP Request - Transaction ID 0xd534d734
4	0.032551	10.0.0.100	10.0.1.1	DHCP	DHCP ACK - Transaction ID 0xd534d734

Tabela 4.10: Captura na subrede do cliente

No.	Tempo	Origem	Destino	Proto	Info
1	0.000000	0.0.0.0	255.255.255.255	DHCP	DHCP Discover - Transaction ID 0xd334d534
2	0.019238	10.0.1.1	10.0.1.10	DHCP	DHCP Offer - Transaction ID 0xd334d534
3	0.020404	10.0.0.0	255.255.255.255	DHCP	DHCP Request - Transaction ID 0xd534d734
4	0.035175	10.0.1.1	10.0.1.10	DHCP	DHCP ACK - Transaction ID 0xd534d734

De maneira similar ao protocolo HTTP via proxy, neste caso há uma forte ligação entre o ambiente e o componente a ser testado. Ao excluir-se o contexto, o ambiente consegue manter comunicação direta com o componente a ser testado – o servidor.

4.5 Discussão

Os resultados encontrados na análise de protocolos reais levam à conclusão de que existem condições ideais sob as quais pode-se efetuar teste de conformidade em contexto. A Tabela 4.11 sumariza os resultados encontrados. A justificativa para explicar a viabilidade ou não de se aplicar teste de conformidade encontra-se em teorias de engenharia de software, em um artigo sobre desenvolvimento estruturado. Dois conceitos importantes associados ao desenvolvimento de sistemas complexos são coesão e acoplamento [Stevens et al., 1999][2]. Acoplamento é uma medida da interdependência entre diferentes módulos[3] e coesão é uma medida da ligação existente entre os elementos dentro de um mesmo módulo. Para ser classificado como bem desenvolvido, um sistema procedural tem que ter baixo acoplamento em termos de haver

[2]Os conceitos de *coesão* e *acoplamento*, aqui utilizados para justificar as condições favoráveis para se executar teste de conformidade em contexto, são os desenvolvidos neste artigo clássico de desenvolvimento estruturado para o paradigma imperativo das linguagens procedurais. Os conceitos foram espandidos e extendidos no paradigma de orientação a objetos. A discussão mais recente encontrada a respeito de desenvolvimento estruturado aconteceu em um painel da conferência OOPSLA'05 (*Object-Oriented Programming, Systems, Languages, and Applications* [Constantine et al., 2005]), da qual participou Larry Constantine, um dos co-autores do artigo clássico.

[3]Nas discussões doravante a palavra *módulo* será utilizada como significado de subsistema, em concordância com o artigo de [Stevens et al., 1999].

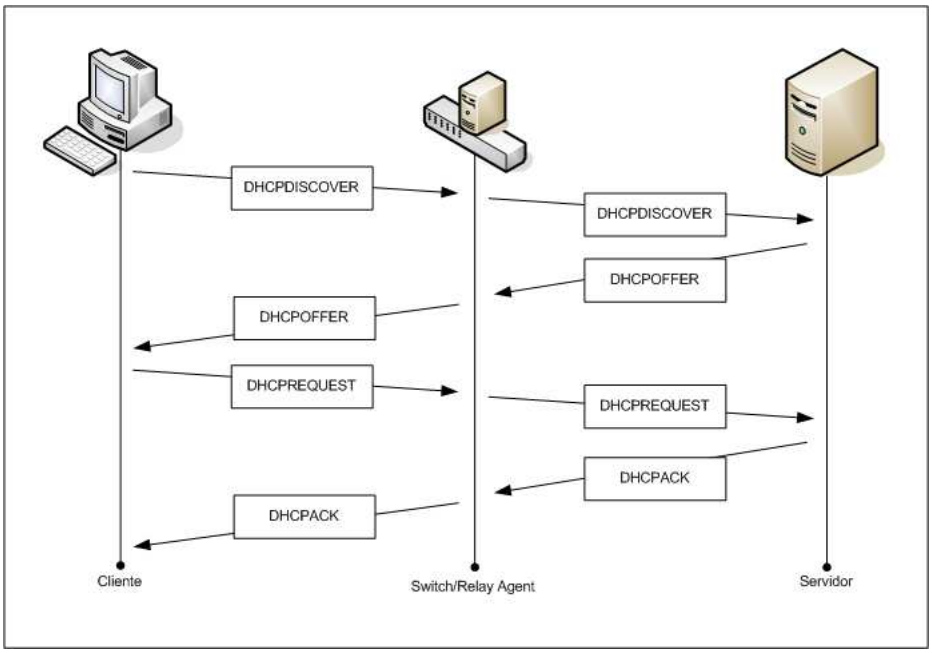


Figura 4.6: Seqüências de mensagens do protocolo DHCP com *relay-agent*

o mínimo de interdependência entre os módulos e alta coesão em termos de haver forte ligação entre os elementos de um módulo individual [Eder et al., 1992].

Tabela 4.11: Sumário da viabilidade da aplicação de teste em contexto

Arquitetura	**Viabilidade**
Pilha do TCP/IP	Não
HTTP via Proxy	Sim
HTTPS via Proxy	Sim / Não
DHCP via Relay	Sim

De fato, a teoria de desenvolvimento estruturado responde às questões encontradas em teste de conformidade em contexto. As condições necessárias e suficientes para que seja possível efetuá-lo são: i) individualmente o componente e o contexto não serem altamente coesos e ii) entre si serem fortemente acoplados. Segundo [Stevens et al., 1999], "o acoplamento aumenta com o aumento da complexidade ou da obscuridade da interface [...] O acoplamento é mais fraco com conexões de dados do que com conexões de controle, que por sua vez é mais fraco que conexões híbridas. A contribuição destes fatores está sumarizada na Tabela 4.12".

Existem padrões estruturais de desenvolvimento cujas aplicações que os implementam

permitem um teste completo de um componente a partir do contexto. Estes padrões são do tipo procurador, adaptador ou decoradores [Gamma et al., 1995] e encontram-se descritos no Apêndice A. O que os caracteriza é justamente o alto acoplamento. Por exemplo, uma classe decorada (considerada como um componente para teste em contexto) pode ser plenamente testada por meio do seu decorador (o contexto). Isto explica os casos em que foi possível efetuar teste de conformidade em contexto em protocolos reais (Tabela 4.11), pois nestes casos o contexto agia como um decorador ou um procurador do componente.

Tabela 4.12: Fatores contribuintes de acoplamento

	Complexidade da Interface	*Tipo de Conexão*	*Tipo de Comunicação*
baixo	simples, óbvia	para módulo por nome	dados
ACOPLAMENTO			controle
alto	complicada, obscura	para elementos internos	híbrido

A inviabilidade de executar teste em contexto na pilha de protocolos do TCP/IP deve-se ao fato de cada camada ser altamente coesa, pois cada uma foi desenvolvida para tratar de tipos distintos de dados, e entre si elas são fracamente acopladas, pois o que as liga é o serviço que a camada inferior oferece à camada imediatamente superior. O caso da falha da comunicação HTTPS via Proxy transparente deve-se ao fato de haver uma alta coesão do componente (o servidor HTTPS) e um fraco e até nulo acoplamento entre o componente e o contexto. O caso positivo de teste em contexto com Proxy HTTP deve-se ao fato do contexto e do componente serem fortemente acoplados. Este mesmo raciocínio aplica-se para o protocolo DHCP via agente de entrega.

Ainda no âmbito de desenvolvimento estruturado [Stevens et al., 1999], um conceito também útil para explicar teste em contexto é o de módulos previsíveis. Um módulo previsível ou bem-comportado é aquele que, quando dado um conjunto de entradas idênticas, opera de maneira idêntica cada vez que é chamado, e o faz de maneira independente de seu ambiente.

A teoria de desenvolvimento estruturado mostra que mesmo um módulo livre de erros pode ser imprevisível. Um exemplo encontrado em [Stevens et al., 1999] é um módulo oscilador que retorna alternativamente zero e um dependendo de quando é chamado. Este módulo

poderia ser utilizado para facilitar dupla buferização. Ele poderia ter vários usuários, onde cada um seria solicitado a chamá-lo um número par de vezes antes de liberar seu controle. Caso alguns dos usuários tivessem um erro que prevenisse um número par de chamadas, todos os outros usuários falhariam. A operação do módulo dadas as mesmas entradas não é constante, o que resulta que o módulo é imprevisível apesar de ser livre de erros.

No âmbito de teste em contexto, o componente sob teste poderia se comportar de maneira imprevisível a partir de um contexto, mesmo o componente sendo livre de falhas e sendo representado por uma FSM determinista. Pode-se considerar o servidor DHCP no contexto de um agente de entrega como um oscilador: um cliente DHCP nunca sabe o endereço IP que receberá do servidor ou mesmo se receberá. Em recebendo, não tem como prever se conseguirá renová-lo. Enfim, além de baixa coesão e forte acoplamento, as evidências sugerem que para ser possível efetuar teste em contexto faz-se necessário que o componente embarcado seja previsível a partir do contexto.

Capítulo 5

Teste de Conformidade em Contexto Guiado por Casos de Teste do Componente

Partindo do princípio de que é possível efetuar teste a partir do contexto, conforme é alegado pelos algoritmos propostos na literatura e vistos no capítulo anterior, será apresentada neste capítulo uma nova abordagem para o problema. A conjectura apresentada é que, se é possível testar um componente a partir do contexto, e o componente foi especificado para comunicar-se com o contexto e vice-versa, então é possível mapear seqüências de teste do componente para o contexto. A hipótese aparentemente é verídica. Caso não fosse, os resultados publicados na literatura não seriam. Utilizando a idéia de mapeamento de casos de teste do componente para o contexto, serão apresentados neste capítulo o algoritmo e alguns conceitos correlacionados. Ele utiliza alguns conceitos dos algoritmos anteriores, com o diferencial de que a busca da seqüência no contexto será guiada por casos de teste do componente. Esta estratégia evita o esforço despendido pelos algoritmos anteriores.

5.1 Justificativa

No modelo de referência ISO OSI, dados passados pelo usuário à camada de aplicação do sistema receberam a denominação de SDU (*Service Data Unit*). A camada de aplicação, então, acresce à SDU (que são os dados do usuário) um cabeçalho chamado PCI (*Protocol Control Information*). O objetivo final desta união é gerar o PDU (*Protocol Data Unit*), correspondente à unidade de dados especificada de um protocolo da camada em questão. Mas, para que possamos realizar serviços entre as camadas, necessitamos das primitivas de serviços, que são, na verdade, informações trocadas entre camadas vizinhas. O modelo OSI possui quatro tipos de primitivas:

- Pedido (request): utilizada para solicitar ou ativar determinado serviço;
- Indicação (indication): informa a ocorrência de determinado evento;
- Resposta (response): utilizada para responder a um determinado evento;
- Confirmação (confirmation): utilizada para confirmar a execução de um serviço solicitado.

Quem administra a negociação entre as primitivas é o protocolo e existem dois tipos de serviços:serviços confirmados e serviços não confirmados. Uma conexão é um serviço confirmado e uma desconexão é um serviço não confirmado. Abaixo é descrita a composição dos dois tipos de serviços:

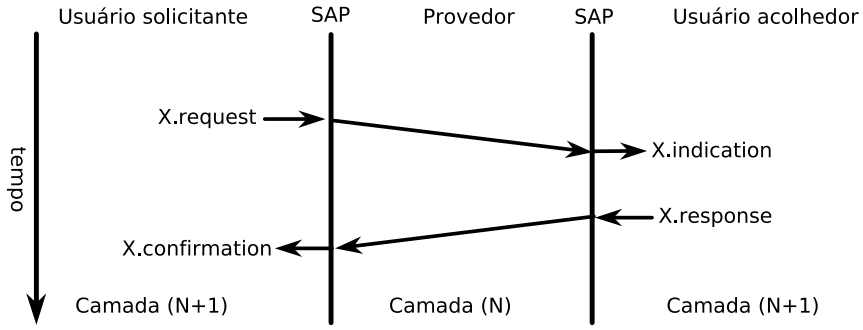


Figura 5.1: Serviço confirmado

- Serviço confirmado: pedido, indicação, resposta e confirmação;
- Serviço não confirmado (desconexão): pedido e indicação.

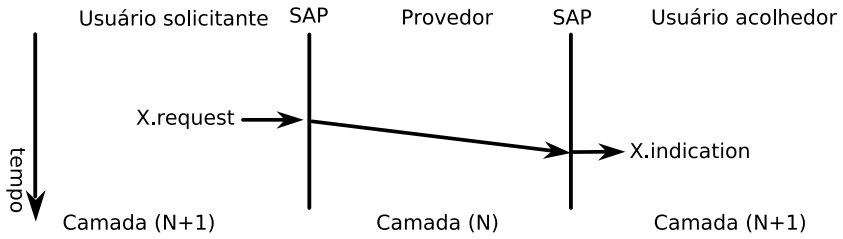


Figura 5.2: Serviço não confirmado

Para ilustrar melhor este modo, será utilizado um exemplo de uma conexão via linha telefônica, mostrado na Figura 5.1:

1. **Connection.Request:** solicita o estabelecimento de uma conexão.
2. **Connection.Indication:** informa à parte chamada.
3. **Connection.Response:** entidade chamada aceita ou rejeita as chamadas.
4. **Connection.Confirmation:** indica ao solicitante se a chamada foi aceita.

Em contraposição, a desconexão da chamada acima é um serviço não confirmado, conforme é ilustrado na Figura 5.2:

1. **Data.Request:** solicita a transmissão de dados;
2. **Data.Indication:** avisa sobre a chegada de dados;
3. **Data.Request:** solicita que a conexão seja liberada;
4. **Data.Indication:** informa ao emissor sobre o pedido.

5.1.1 Serviços e protocolos

É necessário que haja uma clara distinção entre serviços e protocolos. Um serviço é um conjunto de primitivas que uma camada utiliza para comunicar-se com a camada superior imediata. Isto quer dizer que um serviço é uma interface entre duas camadas, onde, como vimos, a inferior age como provedora dos serviços e a superior como usuária do serviço. A função do serviço será dividir as operações que a camada está preparada para realizar, sem implementar o serviço.

Por sua vez, um protocolo é formado por um conjunto de regras que determinam o formato e significado dos pacotes trocados entre as camadas de mesmo nível dos sistemas que estão se comunicando. A ação dos protocolos é transparente ao usuário, ou seja, o usuário não percebe a realização de todos estes procedimentos.

No esquema em camadas do modelo OSI, os protocolos relacionam sistemas em níveis compatíveis - a camada de transporte do sistema A se relaciona com a camada de transporte do sistema B - enquanto os serviços relacionam camadas adjacentes dentro do mesmo sistema (a camada de aplicação se relaciona por serviços com a camada de apresentação, ambas no sistema A, ocorrendo o mesmo no sistema B). É importante observar que serviços não comunicam camadas entre os sistemas A e B.

As camadas adjacentes utilizam o Ponto de Acesso a Serviço (SAP [1]) que é uma interface lógica - software e não hardware - entre as entidades e as camadas adjacentes. Para

[1] do Inglês *Service Access Point*.

que camadas compatíveis se comuniquem há uma associação que recebe o nome de conexão e utiliza o protocolo existente para aquele serviço prestado naquela camada.

5.2 Princípios do algoritmo

O modelo de protocolos e serviços do padrão OSI apresentados na seção anterior e os protocolos para Internet apresentados em capítulos anteriores ilustram que existem dependências [2] entre os sinais de um mesmo protocolo e entre protocolos de módulos que se comunicam. Em razão da existência destes dois tipos de dependência, assumimos que seja possível efetuar teste em contexto por meio de mapeamento de casos de teste do componente. Os passos deste mapeamento encontram-se descritos no Algoritmo 8.

Algoritmo 8 Mapeamento de casos de teste do componente ao contexto

Entrada: T: FSM; C: caso de teste do componente
Saída: U

1: $\lambda_T^{-1} \leftarrow inverte(\lambda_T)$
2: $s \leftarrow s_0$
3: $U \leftarrow NULL$
4: **enquanto** $C \neq NULL$ **faça**
5: $\quad a \leftarrow desenfileira(C)$
6: $\quad k \leftarrow procura(a, T, s)$
7: $\quad enfileira(U, k)$
8: $\quad s \leftarrow proximoEstado(s, T, k)$
9: **fim enquanto**

Inicialmente, vamos definir as funções utilizadas no algoritmo. A função $inverte(\lambda_T)$ gera um mapeamento inverso das transições, isto é, para cada estado do contexto (T), ela descreve que sinal de entrada é utilizado para produzir determinado sinal de saída. A inversão da função λ_T produzirá uma relação quando, para um dado estado, houver mais de uma transição que gere um mesmo sinal de saída. Caso exista somente uma transição que gere um sinal de saída, temos uma função inversa. Caso a transição não produza nenhum sinal de saída, consideraremos um sinal NULL como uma saída especial, aplicando as idéias anteriores a ele.

A função $desenfileira(C)$ pega o próximo sinal de entrada da seqüência de teste C. A função $procura(a, T, s)$ é uma busca em largura que tem como parâmetros o sinal de entrada

[2] que também poderiam ser representadas como uma ordem causal ou ordem de precedência entre sinais.

esperado pelo componente, a especificação do contexto e o estado (ou configuração) onde desejamos iniciar o contexto. Esta função devolve a seqüência de sinais de entrada que devemos aplicar ao contexto para estimular o sinal de entrada a esperado pelo componente. Por último, a função $proximoEstado(s, T, k)$ informa em que estado o contexto se encontrará após gerar a seqüência k.

O algoritmo tem como parâmetros de entrada a especificação do contexto (T) e um caso de teste C gerado a partir do componente P (C será considera como uma fila). Primeiro produzimos a relação λ_T^{-1} (linha 1). Posteriormente iniciamos o contexto em um determinado estado ou configuração (linha 2). Para efeito prático, consideraremos que este é o estado inicial. Caso não seja o estado inicial, será um estado no contexto que produza sinais internos que estimulem transições do estado inicial do componente.

Armazenamos numa fila U o caso de teste produzido pelo algoritmo. De início, U não possui nenhum sinal (linha 3). Em seguida, entramos na iteração do algoritmo (linha 4), que será executada até que a fila C não possua nenhum elemento. Pegamos o próximo elemento da fila (linha 5). Isto é, identificamos qual é o sinal atual esperado pelo componente. Após pesquisamos no estado atual s de T se existe uma transição que produza o sinal de entrada esperado pelo caso de teste C (linha 6). Existem duas possibilidades. Primeiro, não existe uma transição partindo de s que produza um sinal a esperado pelo componente P. Segundo, existe uma transição que produz o sinal a. Este dois casos serão mais detalhados nas Seções 5.2.1 e 5.2.2.

Em seguida inserimos na fila U a seqüência k de sinais estimulados em T para produzir o sinal a(linha 7). Esta mesma seqüência é utilizada para estimular o contexto a alcançar um novo estado s (linha 8). Então, continuamos o laço até que todo o caso de teste seja percorrido.

5.2.1 Não existência de transição

Neste caso temos que procurar um estado alcançável a partir de s que produza a entrada a esperada por P. Eventualmente esta busca conduzirá a caminhos que possuirão sinais internos diferentes de a. Todos estes deverão ser descartados. Se não houver nenhum caminho a partir de s que não inclua sinais internos diferentes de a e que chegue a um estado que produza o sinal a esperado pelo componente P, o algoritmo aborta. Isto significa que a seqüência de teste C que desejávamos exercitar no componente não é testável a partir do contexto. Note que neste caso é possível que haja "deadlock" na interação componente e contexto.

5.2.2 Existência de transição

Este caso é o mais simples de ser tratado. Sabemos de antemão que existe uma transição no estado atual s do contexto que produza a saída esperada pelo componente. No pior das hipóteses existirá mais de uma transição. Caso exista somente uma transição, exercitamos esta transição e continuamos o algoritmo analisando o próximo sinal do caso de teste C.

Caso contrário, isto é, existe mais de uma transição em s que produza o sinal a esperado pelo componente, o algoritmo decide que transições serão descartadas. Esta decisão só pode ser tomada se analisarmos o próximo sinal da seqüência de teste. Eventualmente a análise do próximo sinal da seqüência de teste nos conduzirá a outra tomada de decisão. Conceitualmente isto pode conduzir a uma árvore de decisões. Em casos práticos não ocorrem muitas decisões, tendo em vista que em protocolos reais sempre existe uma dependência entre o sinal atual e o próximo sinal.

Capítulo 6

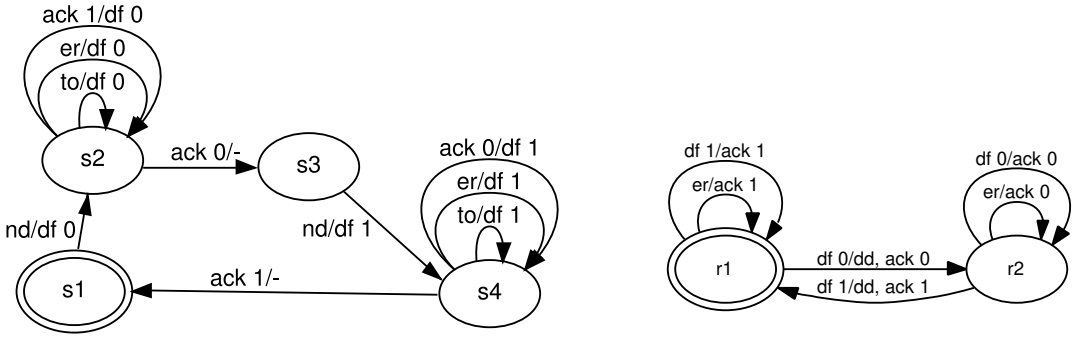
Experimentos

Este capítulo apresenta experimentos com os algoritmos existentes na literatura para teste em contexto, aplicados a protocolos que atendem aos requisitos levantados no capítulo anterior: ser orientado a estados e manipular o mesmo tipo de dados ou ter o mesmo formato de mensagem. Os protocolos escolhidos foram: o ABP (Protocolo do Bit Alternante), que é um protocolo elementar para camada de transporte, cuja função inclui a entrega garantida de mensagens da camada de aplicação no destino; o SCU (Unidade de Controle do Assinante), que é um exemplo copiado de um artigo existente na literatura, que descreve a interação entre um assinante e sua unidade de controle; e o TCP (Protocolo de Controle de Transmissão da Internet), cuja função é fornecer uma comunicação lógica entre processos de aplicações que executam em hospedeiros distintos.

6.1 Protocolo do Bit Alternante (ABP)

O protocolo do bit alternante (ABP) é o mais elementar para transferência confiável de dados sobre um canal susceptível a perdas e a erros de bits. Ele é constituído de duas entidades – o transmissor e o receptor – cujas máquinas encontram-se ilustradas nas Figuras 6.1(a) e 6.1(b). A transmissão de dados é feita de maneira unidirecional do transmissor ao receptor. O transmissor foi considerado o contexto e o receptor o componente.

O primeiro experimento feito com o ABP foi compor as máquinas conforme os passos do Algoritmo 5 (Capítulo 3. A composição encontra-se ilustrada na Figura 6.2 (página 61). Em função da enorme quantidade de transições e de estados gerados, deduz-se que a composição não é um método eficiente de se aplicar a teste de conformidade em contexto. Portanto, a geração de casos de teste para a máquina composta será desconsiderada nos experimentos.



(a) Transmissor (b) Receptor

Figura 6.1: FSMs do Protocolo do Bit Alternante

De maneira similar não será considerada a geração de casos de teste pelo algoritmo *Hit-or-Jump* (Algoritmo 6, Capítulo 3), levando-se em consideração que o algoritmo foi desenvolvido para uma ferramenta específica (Object Geode) e ainda que sua heurística não condiz com protocolos de uso real. A idéia de retornar a um estado anterior na máquina do contexto em tese poderia indicar o envio de um sinal de *reset*, uma vez que não é incomum que não haja transições para retornar ao estado anterior, como pode ser observado nas máquinas de estados do TCP e do cliente DHCP.

Os algoritmos utilizados nos experimentos com o protocolo ABP foram:

- Passeio de transições no transmissor e avaliação de sua cobertura no receptor, ilustrados na Tabela 6.1;
- Passeio de transições no receptor e seu mapeamento ao transmissor, ilustrados na Tabela 6.2;
- Busca em profundidade no transmissor e avaliação de sua cobertura no receptor, ilustradas na Tabela 6.3;
- Busca em profundidade no receptor e seu mapeamento ao transmissor, ilustradas na Tabela 6.4.

De acordo com o que pode ser observado nas tabelas, o processo de mapeamento de sinais é eficiente quando se compara o passeio de transições gerado no contexto (Tabela 6.1) com o passeio gerado no componente e mapeado ao contexto (Tabela 6.2). O mesmo não se pode afirmar para a busca em profundidade – tanto a busca gerada no contexto (Tabela 6.3) quanto a busca mapeada do componente ao contexto (Tabela 6.4) apresentaram a mesma cobertura de falhas. Em outras palavras, no caso da busca em profundidade o processo de mapeamento traria

uma sobrecarga a mais do que a execução da busca em profundidade diretamente no contexto. Os próximos experimentos devem elucidar melhor estes resultados.

No tocante às particularidades do protocolo ABP, as transições relacionadas a erros, temporizadores e mensagens duplicadas são impossíveis ou pouco prováveis de serem estimuladas a partir do ambiente. Mais da metade das transições do transmissor listadas na Tabela 6.1 recaem neste caso (segunda, terceira, sétima, oitava, nona e décima transições). Quatro das seis transições do receptor na Tabela 6.2 recaem neste caso. O que se pode observar pelos casos de teste gerados por busca em profundidade é que estas transições são evitadas, isto é, elas tendem a não ser inclusas nas seqüências de teste.

Tabela 6.1: Passeio de transições no contexto do protocolo ABP

Transmissor	**Receptor**
s1: $\xrightarrow[\text{data frame 0}]{\text{new data}}$ s2	r1: $\xrightarrow[\text{delivery data, acknowledgement 0}]{\text{data frame 0}}$ r2
s2: $\xrightarrow[\text{data frame 0}]{\text{time-out}}$ s2	r1: $\xrightarrow[\text{delivery data, acknowledgement 0}]{\text{data frame 0}}$ r2
s2: $\xrightarrow[\text{data frame 0}]{\text{error}}$ s2	r1: $\xrightarrow[\text{delivery data, acknowledgement 0}]{\text{data frame 0}}$ r2
s2: $\xrightarrow[\text{data frame 0}]{\text{acknowledgement 1}}$ s2	r1: $\xrightarrow[\text{delivery data, acknowledgement 0}]{\text{data frame 0}}$ r2
s2: $\xrightarrow{\text{acknowledgement 0}}$ s3	
s3: $\xrightarrow[\text{data frame 1}]{\text{new data}}$ s4	r2: $\xrightarrow[\text{delivery data, acknowledgement 1}]{\text{data frame 1}}$ r1
s4: $\xrightarrow[\text{data frame 1}]{\text{time-out}}$ s4	r2: $\xrightarrow[\text{delivery data, acknowledgement 1}]{\text{data frame 1}}$ r1
s4: $\xrightarrow[\text{data frame 1}]{\text{error}}$ s4	r2: $\xrightarrow[\text{delivery data, acknowledgement 1}]{\text{data frame 1}}$ r1
s4: $\xrightarrow[\text{data frame 1}]{\text{acknowledgement 0}}$ s4	
s4: $\xrightarrow{\text{acknowledgement 1}}$ s1	

6.2 Unidade de Controle do Assinante (SCU)

O próximo experimento foi feito com base em um exemplo de teste em contexto encontrado na literatura [Lima and Cavalli, 1997]: a unidade de controle do assinante. O modelo representa um sistema de telecomunicações composto de duas máquinas – Assinante (*Subscriber*) e a Unidade de Controle do Assinante (*SCU - Subscriber Control Unit*) – cujos comportamentos encontram-se representados, respectivamente, nas Figuras 6.4 e 6.3. A SCU é o contexto e o *Subscriber* o componente.

Tabela 6.2: Mapeamento de um passeio de transições no componente do ABP

Receptor	**Transmissor**
r1: $\xrightarrow[\text{delivery data, acknowledgement 0}]{\text{data frame 0}}$ r2	s1: $\xrightarrow[\text{data frame 0}]{\text{new data}}$ s2
r2: $\xrightarrow[\text{acknowledgement 0}]{\text{error}}$ r2	
r2: $\xrightarrow[\text{acknowledgement 0}]{\text{data frame 0}}$ r2	s2: $\xrightarrow[\text{data frame 0}]{\text{time-out}}$ s3
r2: $\xrightarrow[\text{delivery data, acknowledgement 1}]{\text{data frame 1}}$ r1	s3: $\xrightarrow[\text{data frame 1}]{\text{new data}}$ s4
r1: $\xrightarrow[\text{acknowledgement 1}]{\text{error}}$ r1	
r1: $\xrightarrow[\text{acknowledgement 1}]{\text{data frame 1}}$ r1	s4: $\xrightarrow[\text{data frame 1}]{\text{time-out}}$ s1

Tabela 6.3: Busca em profundidade no contexto do protocolo ABP

Transmissor	**Receptor**
s1: $\xrightarrow[\text{data frame 0}]{\text{new data}}$ s2	r1: $\xrightarrow[\text{delivery data, acknowledgement 0}]{\text{data frame 0}}$ r2
s2: $\xrightarrow{\text{acknowledgement 0}}$ s3	
s3: $\xrightarrow[\text{data frame 1}]{\text{new data}}$ s4	r2: $\xrightarrow[\text{delivery data, acknowledgement 1}]{\text{data frame 1}}$ r1
s4: $\xrightarrow{\text{acknowledgement 1}}$ s1	

Tabela 6.4: Mapeamento de uma busca em profundidade no componente do ABP

Receptor	**Transmissor**
r1: $\xrightarrow[\text{delivery data, acknowledgement 0}]{\text{data frame 0}}$ r2	s1: $\xrightarrow[\text{data frame 0}]{\text{new data}}$ s2
	s2: $\xrightarrow{\text{acknowledgement 0}}$ s3
r2: $\xrightarrow[\text{delivery data, acknowledgement 1}]{\text{data frame 1}}$ r1	s3: $\xrightarrow[\text{data frame 1}]{\text{new data}}$ s4
	s4: $\xrightarrow{\text{acknowledgement 1}}$ s1

De maneira similar à seção anterior, foram executadas quatros experimentos neste sistema, a saber:

- Passeio de transições no SCU e avaliação de sua cobertura no assinante, ilustrados na Tabela 6.5;
- Passeio de transições no assinante e seu mapeamento ao SCU, ilustrados na Tabela 6.6;
- Busca em profundidade no SCU e avaliação de sua cobertura no assinante, ilustradas na Tabela 6.7;
- Busca em profundidade no assinante e seu mapeamento ao SCU, ilustradas na Tabela 6.8.

De maneira similar aos resultados da seção anterior, o processo de mapeamento de sinais é eficiente quando se compara o passeio de transições gerado no contexto (Tabela 6.5) com o passeio gerado no componente e mapeado ao contexto (Tabela 6.6). No caso das buscas em profundidade a busca gerada no contexto (Tabela 6.7) e a busca mapeada do componente ao contexto (Tabela 6.8) apresentaram a mesma cobertura de falhas. Estes resultados reforçam as suposições levantadas a partir da análise na seção anterior, isto é, o processo de mapeamento é eficiente, mas traz sobrecarga quando uma heurística de busca em profundidade é utilizada.

No tocante às particularidades do protocolo SCU, de maneira análoga ao protocolo ABP, as transições relacionadas a erros, temporizadores e mensagens duplicadas são impossíveis ou pouco prováveis de serem estimuladas a partir do ambiente. Proporcionalmente, comparado ao protocolo ABP há menos transições deste gênero no protocolo SCU. Adicionalmente, no protocolo há transições não deterministas, a saber, aquelas da unidade de controle (SCU) que manipulam uma chamada entrante (*call_arriving*).

6.3 Protocolo de Controle de Transmissão (TCP)

O último protocolo utilizado nos experimentos foi o protocolo TCP. Bem mais complexo que o ABP, o TCP provê o serviço de transporte com possibilidade de fluxo bidirecional entre duas aplicações. Ele é um dos protocolos de transporte que fica acima da camada interredes do modelo TCP/IP, utilizado na Internet. Foi desenvolvido para permitir que duas pontas, uma na origem e outra no destino, conduzam uma conversação. Portanto, utilizado para uma comunicação fim-a-fim. É um protocolo orientado a conexão confiável o qual permite que um fluxo de dados originado em uma máquina seja entregue sem erros a uma outra máquina na Internet. Ele fragmenta o fluxo de entrada de bytes em mensagens discretas e passa uma a uma

Tabela 6.5: Passeio de transições no contexto do protocolo SCU

SCU	**Subscriber**
idle: $\xrightarrow{call\_arriving}$ idle	
idle: $\xrightarrow[call]{call\_arriving}$ wait_for_answer	idle: $\xrightarrow{call}$ ringing
	ringing: $\xrightarrow[response]{off\_hook}$ wait_for_stop_ringing
wait_for_answer: $\xrightarrow[stop\_ringing]{response}$ conversation	wait_for_stop_ringing: $\xrightarrow{stop\_ringing}$ conversation
conversation: $\xrightarrow[busy]{release}$ fault	conversation: $\xrightarrow{busy}$ fault
fault: $\xrightarrow{hang\_up}$ idle	fault: $\xrightarrow[hang\_up]{hang\_up}$ idle
idle: $\xrightarrow[call]{call\_arriving}$ wait_for_answer	idle: $\xrightarrow{call}$ ringing
	ringing: $\xrightarrow[response]{off\_hook}$ wait_for_stop_ringing
wait_for_answer: $\xrightarrow[stop\_ringing]{response}$ conversation	wait_for_stop_ringing: $\xrightarrow{stop\_ringing}$ conversation
	conversation: $\xrightarrow[hang\_up]{hang\_up}$ control_by_called
conversation: $\xrightarrow{hang\_up}$ control_by_called	
	control_by_called: $\xrightarrow[response]{off\_hook}$ conversation
control_by_called: $\xrightarrow{response}$ conversation	conversation: $\xrightarrow[hang\_up]{hang\_up}$ control_by_called
conversation: $\xrightarrow{hang\_up}$ control_by_called	control_by_called: $\xrightarrow{release}$ idle
control_by_called: $\xrightarrow[release]{timeout-hang\_up}$ idle	idle: $\xrightarrow{call}$ ringing
idle: $\xrightarrow[call]{call\_arriving}$ wait_for_answer	ringing: $\xrightarrow[response]{off\_hook}$ wait_for_stop_ringing
	wait_for_stop_ringing: $\xrightarrow{stop\_ringing}$ conversation
wait_for_answer: $\xrightarrow[stop\_ringing]{response}$ conversation	conversation: $\xrightarrow[hang\_up]{hang\_up}$ control_by_called
conversation: $\xrightarrow{hang\_up}$ control_by_called	control_by_called: $\xrightarrow{release}$ idle
control_by_called: $\xrightarrow[release]{release}$ idle	
idle: $\xrightarrow[call]{call\_arriving}$ wait_for_answer	
wait_for_answer: $\xrightarrow[stop\_ringing]{ringing\_timer}$ idle	
idle: $\xrightarrow[call]{call\_arriving}$ wait_for_answer	
wait_for_answer: $\xrightarrow[stop\_ringing]{release}$ idle	

Tabela 6.6: Mapeamento de um passeio de transições no componente do SCU

Subscriber	**SCU**
idle: $\xrightarrow{call}$ ringing	idle: $\xrightarrow[call]{call\_arriving}$ wait_for_answer
ringing: $\xrightarrow[response]{off\_hook}$ wait_for_stop_ringing	
	wait_for_answer: $\xrightarrow[stop\_ringing]{response}$ conversation
wait_for_stop_ringing: $\xrightarrow{stop\_ringing}$ conversation	conversation: $\xrightarrow[busy]{release}$ fault
conversation: $\xrightarrow{busy}$ fault	
fault: $\xrightarrow[hang\_up]{hang\_up}$ idle	
	fault: $\xrightarrow{hang\_up}$ idle
idle: $\xrightarrow{call}$ ringing	idle: $\xrightarrow[call]{call\_arriving}$ wait_for_answer
ringing: $\xrightarrow[response]{off\_hook}$ wait_for_stop_ringing	
wait_for_stop_ringing: $\xrightarrow{stop\_ringing}$ conversation	wait_for_answer: $\xrightarrow[stop\_ringing]{response}$ conversation
conversation: $\xrightarrow[hang\_up]{hang\_up}$ control_by_called	
	conversation: $\xrightarrow{hang\_up}$ control_by_called
control_by_called: $\xrightarrow[response]{off\_hook}$ conversation	
	control_by_called: $\xrightarrow{response}$ conversation
conversation: $\xrightarrow[hang\_up]{hang\_up}$ control_by_called	
	conversation: $\xrightarrow{hang\_up}$ control_by_called
control_by_called: $\xrightarrow{release}$ idle	control_by_called: $\xrightarrow[release]{release}$ idle

Tabela 6.7: Busca em profundidade no contexto do protocolo SCU

SCU	**Subscriber**
idle: $\xrightarrow[call]{call\_arriving}$ wait_for_answer	idle: $\xrightarrow{call}$ ringing
	ringing: $\xrightarrow[response]{off\_hook}$ wait_for_stop_ringing
wait_for_answer: $\xrightarrow[stop\_ringing]{response}$ conversation	wait_for_stop_ringing: $\xrightarrow{stop\_ringing}$ conversation
	conversation: $\xrightarrow[hang\_up]{hang\_up}$ control_by_called
conversation: $\xrightarrow{hang\_up}$ control_by_called	control_by_called: $\xrightarrow[response]{off\_hook}$ conversation
control_by_called: $\xrightarrow{response}$ conversation	
conversation: $\xrightarrow[busy]{release}$ fault	conversation: $\xrightarrow{busy}$ fault
	fault: $\xrightarrow[hang\_up]{hang\_up}$ idle
fault: $\xrightarrow{hang\_up}$ idle	

Tabela 6.8: Mapeamento de uma busca em profundidade no componente do SCU

Subscriber	**SCU**
idle: $\xrightarrow{call}$ ringing	idle: $\xrightarrow[call]{call\_arriving}$ wait_for_answer
ringing: $\xrightarrow[response]{off\_hook}$ wait_for_stop_ringing	
wait_for_stop_ringing: $\xrightarrow{stop\_ringing}$ conversation	wait_for_answer: $\xrightarrow[stop\_ringing]{response}$ conversation
conversation: $\xrightarrow[hang\_up]{hang\_up}$ control_by_called	
control_by_called: $\xrightarrow[response]{off\_hook}$ conversation	conversation: $\xrightarrow{hang\_up}$ control_by_called
	control_by_called: $\xrightarrow{response}$ conversation
conversation: $\xrightarrow{busy}$ fault	conversation: $\xrightarrow[busy]{release}$ fault
fault: $\xrightarrow[hang\_up]{hang\_up}$ idle	
	fault: $\xrightarrow{hang\_up}$ idle

na camada de inter-redes. No destino, o processo receptor TCP remonta as mensagens recebidas em um fluxo de saída. O TCP também manipula fluxo de controle para ter certeza que um transmissor rápido não inunde um cliente lento com mais mensagens do que ele pode lidar.

Com toda sua complexidade, a parte de interesse para o experimento com o TCP para teste em contexto é o comportamento de um cliente e de um servidor TCP designados para uma mesma aplicação. Ambos são obtidos a partir da poda de um estado específico e de suas respectivas transições de origem e de destino da FSM ilustrada na Figura 2.10. Dos estados listados na Tabela 6.9 um cliente TCP não implementa o *LISTEN* e um servidor não costuma implementar o *SYN_SENT* [Stevens, 1993]. A FSM que ilustra o comportamento de um cliente TCP encontra-se ilustrado na Figura 6.5 e a do servidor na Figura 6.6.

O experimento considerou o cliente como o contexto e o servidor como o componente a ser testado. Corroborando com os resultados das seções anteriores, o processo de mapeamento de sinais foi eficiente quando se compara o passeio de transições gerado no contexto (Tabela 6.10) com o passeio gerado no componente e mapeado ao contexto (Tabela 6.11). As buscas em profundidade geradas no contexto (Tabela 6.12) e a mapeada do componente ao contexto (Tabela 6.13) apresentaram também a mesma cobertura de falhas. Adicionado aos resultados obtidos nos dois experimentos anteriores, conclui-se que o processo de mapeamento é eficiente em termos da geração dos casos de teste. Seu único inconveniente é o tempo de execução quando a heurística estabelecida para os testes é o de busca em profundidade.

Com relação às particularidades do protocolo TCP, de maneira similar aos protocolos ABP e SCU, as transições relacionadas a erros, temporizadores e mensagens duplicadas são impossíveis ou pouco prováveis de serem estimuladas a partir do ambiente. A diferença principal do TCP comparado com os protocolos anteriores é que o TCP possui a opção de ambos os lados fecharem a conexão. A última coluna das Tabelas 6.10 e 6.11 marca o tipo de fechamento que aconteceu, exceto para o caso do fechamento da maneira convencional. Para algumas poucas aplicações da Internet é possível executar o fechamento pela metade (*half-close*) [Stevens, 1993], que exercita as mesmas transições que o fechamento simultâneo. Em outras palavras, tal processo pode ser estimulado a partir do contexto. Por outro lado, o fechamento iniciado pelo servidor e suas transações seguintes, como o nome sugere, não podem ser estimulados a partir do contexto. Portanto, assim como as transições relacionadas a erros, temporizadores e mensagens duplicadas, as transações ativadas pelo componente são impossíveis ou pouco prováveis de serem exercitadas a partir do contexto.

Tabela 6.9: Descrição dos estados do protocolo TCP

Estado	**Descrição**
CLOSED	Nenhuma conexão está ativa ou pendente
LISTEN	O servidor está à espera de conexão(ões)
SYN RCVD	Uma solicitação de conexão chegou; espera por ACK
SYN SENT	A aplicação iniciou o processo de abertura de uma conexão
ESTABLISHED	O estado normal de transferência de dados
FIN WAIT 1	A aplicação disse que terminou
FIN WAIT 2	O outro lado concordou em liberar a conexão
TIMED WAIT	Aguarda a entrega dos últimos pacotes
CLOSING	Ambos os lados tentaram fechar a conexão simultaneamente
CLOSE WAIT	O outro lado iniciou uma liberação de conexão
LAST ACK	Aguarda a entrega dos últimos pacotes

6.4 Discussão

O resultados obtidos nos experimentos das seções anteriores estão resumidos na Tabela 6.14, que representa a razão entre o número de transições exercitadas no componente e o número de transições exercitadas no contexto, para cada um dos métodos utilizados. A abordagem guiada por casos de teste do componente mostrou-se ser a melhor solução para o problema de teste de conformidade em contexto. Este resultado pode ser deduzido a partir dos passeios gerados na especificação do componente e mapeados ao contexto nos experimentos com os protocolos ABP, SCU e TCP. Se comparados com os passeios gerados a partir da especificação do contexto, observa-se que este tenta exercitar transições que não afetam diretamente o componente. Portanto, teste em contexto guiado por casos de teste do componente exercita menos transições no contexto para uma melhor cobertura de falhas de saída no componente.

Tabela 6.14: Razão entre transições exercitadas no componente e no contexto

Método	**ABP**	**SCU**	**TCP**
Passeio de Transições no Contexto	$7/10$	$17/19$	$21/19$
Passeio de Transições no Componente	$6/4$	$12/10$	$21/17$
Busca em Profundidade no Contexto	$2/4$	$7/6$	$7/6$
Busca em Profundidade no Componente	$2/4$	$7/6$	$7/6$
DS, W-set, UIO	–	–	–

Os métodos descritos no Capítulo 2 não foram levados em consideração nos experimentos porque eles são difíceis de ser aplicados a teste de conformidade e muito mais difíceis de se aplicar a teste de conformidade em contexto. Sua aplicação dependeria da composição das

Tabela 6.10: Passeio de transições no contexto do protocolo TCP

Cliente	**Servidor**	**Tipo de fechamento**
	closed: $\xrightarrow{passiveopen}$ listen	
closed: $\xrightarrow[syn]{activeopen}$ syn_sent	listen: $\xrightarrow[syn+ack]{syn}$ syn_rcvd	
syn_sent: $\xrightarrow[ack]{syn+ack}$ established	syn_rcvd: $\xrightarrow{ack}$ established	
established: $\xrightarrow[fin]{close}$ fin_wait_1	established: $\xrightarrow[fin]{close}$ fin_wait_1	Half-close or simultaneous close
fin_wait_1: $\xrightarrow[ack]{fin}$ closing	fin_wait_1: $\xrightarrow[ack]{fin}$ closing	Half-close or simultaneous close
closing: $\xrightarrow{ack}$ time_wait	closing: $\xrightarrow{ack}$ time_wait	Half-close or simultaneous close
time_wait: $\xrightarrow{timeout}$ closed	time_wait: $\xrightarrow{timeout}$ closed	Half-close or simultaneous close
	closed: $\xrightarrow{passiveopen}$ listen	
closed: $\xrightarrow[syn]{activeopen}$ syn_sent	listen: $\xrightarrow[syn+ack]{syn}$ syn_rcvd	
syn_sent: $\xrightarrow[ack]{syn+ack}$ established	syn_rcvd: $\xrightarrow{ack}$ established	
established: $\xrightarrow[fin]{close}$ fin_wait_1	established: $\xrightarrow[ack]{fin}$ close_wait	
fin_wait_1: $\xrightarrow{ack}$ fin_wait_2	close_wait: $\xrightarrow[fin]{close}$ last_ack	
fin_wait_2: $\xrightarrow[ack]{fin}$ time_wait	last_ack: $\xrightarrow{ack}$ closed	
time_wait: $\xrightarrow{timeout}$ closed	closed: $\xrightarrow{passiveopen}$ listen	
closed: $\xrightarrow[syn]{activeopen}$ syn_sent	listen: $\xrightarrow[syn+ack]{syn}$ syn_rcvd	
syn_sent: $\xrightarrow[ack]{syn+ack}$ established	syn_rcvd: $\xrightarrow{ack}$ established	
	established: $\xrightarrow[fin]{close}$ fin_wait_1	Server active close
established: $\xrightarrow[ack]{fin}$ close_wait	fin_wait_1: $\xrightarrow{ack}$ fin_wait_2	Server active close
close_wait: $\xrightarrow[fin]{close}$ last_ack	fin_wait_2: $\xrightarrow[ack]{fin}$ time_wait	Server active close
last_ack: $\xrightarrow{ack}$ closed	time_wait: $\xrightarrow{timeout}$ closed	Server active close
	closed: $\xrightarrow{passiveopen}$ listen	
closed: $\xrightarrow[syn]{activeopen}$ syn_sent		
syn_sent: $\xrightarrow{close}$ closed		

Tabela 6.11: Mapeamento de um passeio de transições no componente do TCP

Servidor	**Cliente**	**Tipo de fechamento**
closed: $\xrightarrow{passiveopen}$ listen		
listen: $\xrightarrow[syn+ack]{syn}$ syn_rcvd	closed: $\xrightarrow[syn]{activeopen}$ syn_sent	
	syn_sent: $\xrightarrow[ack]{syn+ack}$ established	
syn_rcvd: $\xrightarrow[fin]{close}$ fin_wait_1	established: $\xrightarrow[fin]{close}$ fin_wait_1	Half-close or simultaneous close
fin_wait_1: $\xrightarrow[ack]{fin}$ closing	fin_wait_1: $\xrightarrow[ack]{fin}$ closing	Half-close or simultaneous close
closing: $\xrightarrow{ack}$ time_wait	closing: $\xrightarrow{ack}$ time_wait	Half-close or simultaneous close
time_wait: $\xrightarrow{timeout}$ closed	time_wait: $\xrightarrow{timeout}$ closed	Half-close or simultaneous close
closed: $\xrightarrow{passiveopen}$ listen		
listen: $\xrightarrow[syn+ack]{syn}$ syn_rcvd	closed: $\xrightarrow[syn]{activeopen}$ syn_sent	
syn_rcvd: $\xrightarrow{ack}$ established	syn_sent: $\xrightarrow[ack]{syn+ack}$ established	
established: $\xrightarrow[fin]{close}$ fin_wait_1		Server active close
fin_wait_1: $\xrightarrow{ack}$ fin_wait_2	established: $\xrightarrow[ack]{fin}$ close_wait	Server active close
fin_wait_2: $\xrightarrow[ack]{fin}$ time_wait	close_wait: $\xrightarrow[fin]{close}$ last_ack	Server active close
time_wait: $\xrightarrow{timeout}$ closed	last_ack: $\xrightarrow{ack}$ closed	Server active close
closed: $\xrightarrow{passiveopen}$ listen		
listen: $\xrightarrow[syn+ack]{syn}$ syn_rcvd	closed: $\xrightarrow[syn]{activeopen}$ syn_sent	
syn_rcvd: $\xrightarrow{ack}$ established	syn_sent: $\xrightarrow[ack]{syn+ack}$ established	
established: $\xrightarrow[ack]{fin}$ close_wait	established: $\xrightarrow[fin]{close}$ fin_wait_1	
close_wait: $\xrightarrow[fin]{close}$ last_ack	fin_wait_1: $\xrightarrow{ack}$ fin_wait_2	
last_ack: $\xrightarrow{ack}$ closed	fin_wait_2: $\xrightarrow[ack]{fin}$ time_wait	
closed: $\xrightarrow{passiveopen}$ listen	time_wait: $\xrightarrow{timeout}$ closed	
listen: $\xrightarrow{close}$ closed		

Tabela 6.12: Busca em profundidade no contexto do protocolo TCP

Cliente	**Servidor**
	closed: $\xrightarrow{passiveopen}$ listen
closed: $\xrightarrow[syn]{activeopen}$ syn_sent	listen: $\xrightarrow[syn+ack]{syn}$ syn_rcvd
syn_sent: $\xrightarrow[ack]{syn+ack}$ established	syn_rcvd: $\xrightarrow{ack}$ established
established: $\xrightarrow[fin]{close}$ fin_wait_1	established: $\xrightarrow[ack]{fin}$ close_wait
fin_wait_1: $\xrightarrow{ack}$ fin_wait_2	close_wait: $\xrightarrow[fin]{close}$ last_ack
fin_wait_2: $\xrightarrow[ack]{fin}$ time_wait	last_ack: $\xrightarrow{ack}$ closed
time_wait: $\xrightarrow{timeout}$ closed	closed: $\xrightarrow{passiveopen}$ listen

Tabela 6.13: Mapeamento de uma busca em profundidade no componente do TCP

Servidor	**Cliente**
closed: $\xrightarrow{passiveopen}$ listen	
listen: $\xrightarrow[syn+ack]{syn}$ syn_rcvd	closed: $\xrightarrow[syn]{activeopen}$ syn_sent
syn_rcvd: $\xrightarrow{ack}$ established	syn_sent: $\xrightarrow[ack]{syn+ack}$ established
established: $\xrightarrow[ack]{fin}$ close_wait	established: $\xrightarrow[fin]{close}$ fin_wait_1
close_wait: $\xrightarrow[fin]{close}$ last_ack	fin_wait_1: $\xrightarrow{ack}$ fin_wait_2
last_ack: $\xrightarrow{ack}$ closed	fin_wait_2: $\xrightarrow[ack]{fin}$ time_wait
closed: $\xrightarrow{passiveopen}$ listen	time_wait: $\xrightarrow{timeout}$ closed

máquinas de estado do contexto e do componente, o que é ineficiente.

Os resultados obtidos por meio das seqüências de teste geradas a partir das buscas em profundidade poderiam levar à conclusão de que o mapeamento das seqüências de teste do componente não seria eficiente nestas configurações. No entanto, estas conclusões são parcialmente verídicas, pois as especificações dos protocolos considerados é que levam a tal conclusão. Por exemplo, não seria difícil de imaginar um contexto cuja máquina de estados possua estados não relacionados à comunicação com o componente.

Cite-se como exemplo o protocolo Cotê-de-Resyste [Vries et al., 2000], cujo comportamento encontra-se ilustrado na máquina de estados da Figura 6.7. Supondo-o como um contexto e A, B e C como componentes a serem testados, pode-se afirmar que os estados *Aconnected* e *ABconnected* podem ser visitados sem que haja nenhuma comunicação com o componente C.

A avaliação dos algoritmos existentes para teste em contexto levou à conclusão de que o algoritmo do teste direto é o melhor deles na razão entre complexidade e cobertura de falhas de saída. A busca em profundidade utilizada cobre o funcionamento convencional da interação entre o componente e o contexto. As transições relacionadas ao tratamento de erros, de temporizadores e de mensagens duplicadas são automaticamente descartadas.

O algoritmo de composição das máquinas de estados do componente e do contexto e seus derivados (redução da máquina composta) foram desconsiderados dos experimentos por serem inerentemente ineficientes. O algoritmo *hit-or-jump* foi desconsiderado pela mesma razão. A proposta de escolha aleatória de transições e o retorno para encontrar transações ainda não visitadas não parecem plausíveis.

Portanto, dos algoritmos estudados o do teste direto é o mais eficiente e o algoritmo desenvolvido, guiado por casos de teste de componentes, é o que gera as melhores seqüências de teste, qualquer que seja o critério de cobertura de falhas escolhido.

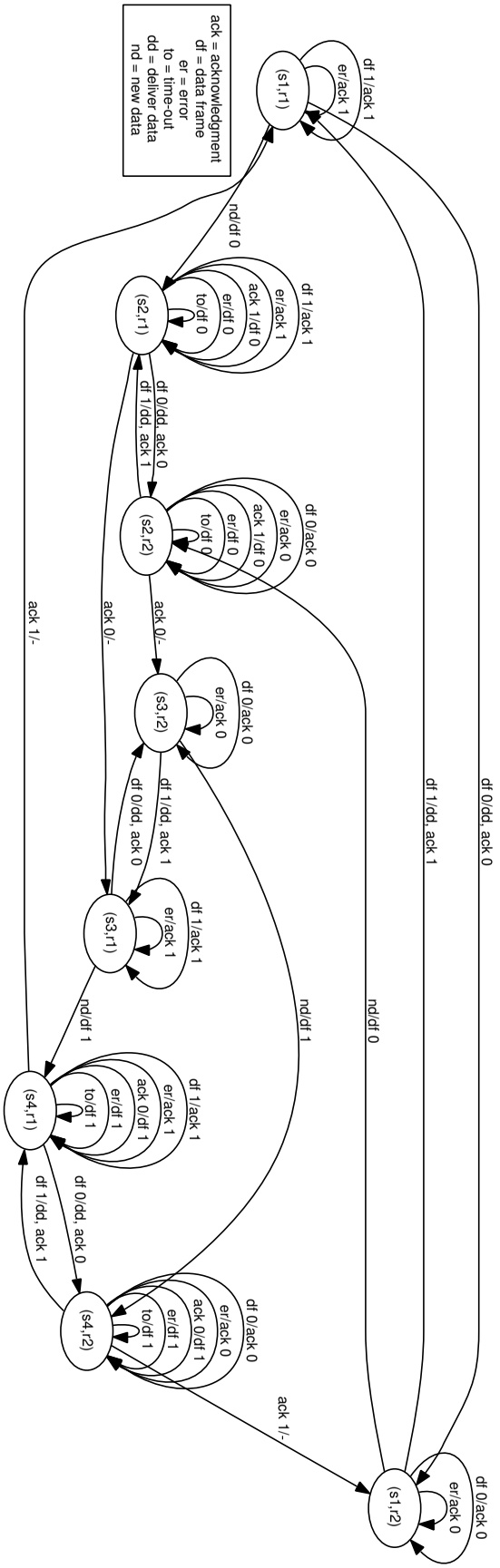


Figura 6.2: FSM composta do Protocolo do Bit Alternante

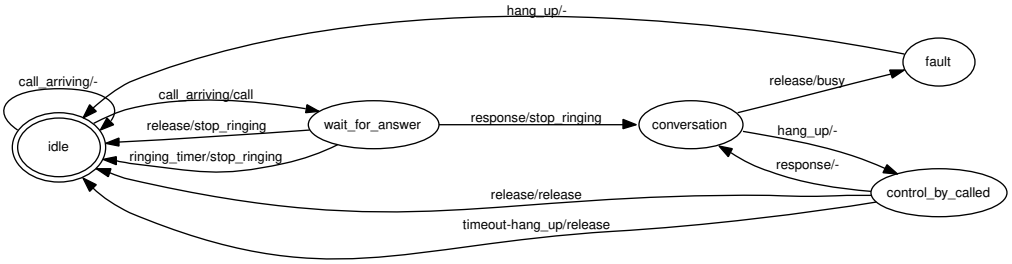


Figura 6.3: Unidade de Controle do Assinante

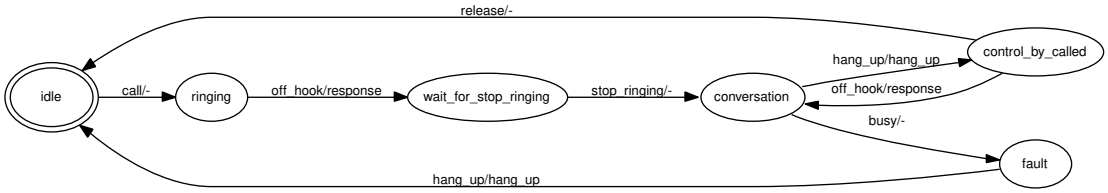


Figura 6.4: Comportamento do Assinante

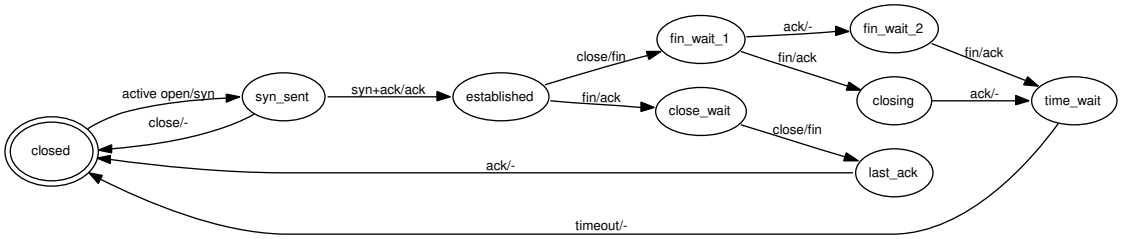


Figura 6.5: Cliente do Protocolo de Controle de Transmissão

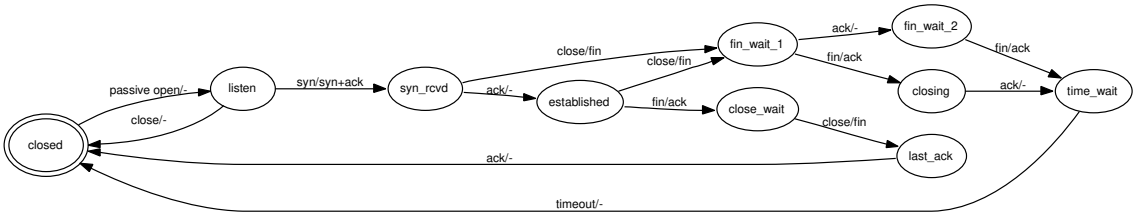


Figura 6.6: Servidor do Protocolo de Controle de Transmissão

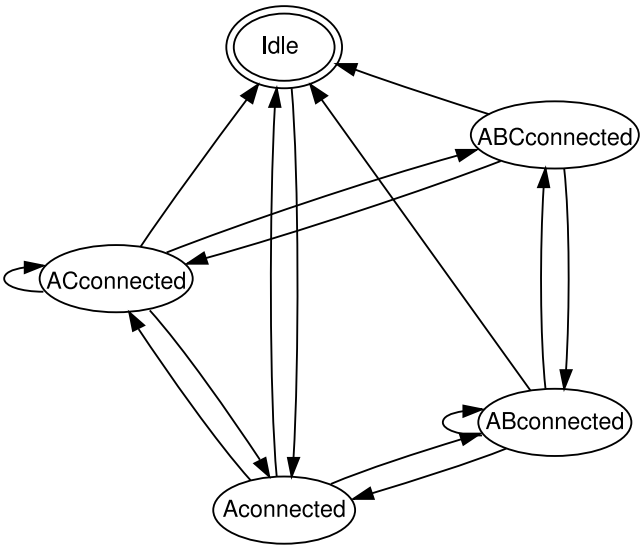


Figura 6.7: Máquina de estados do protocolo Cotê de Resyste

Capítulo 7

Conclusão

O resultados obtidos a partir da análise da execução de protocolos reais mostraram que há condições favoráveis para que se possa fazer teste de conformidade de protocolos em contexto. A primeira e mais importante delas é o acoplamento, que é uma medida de interdependência entre diferentes módulos ou subsistemas. Mesmo o simples acoplamento não é garantia de que se consiga realizá-lo. Os subsistemas das camadas do TCP/IP analisados no Capítulo 4 são acoplados mas foram desenvolvidos para manipular tipos distintos de dados: de sinais elétricos, ondas e bits nas camadas mais inferiores a fragmentos de pacotes e mensagens de aplicação nas camadas superiores. Um forte acoplamento de controle – ter sinais em comum – e de dados – manipular os mesmo tipos de dados – são duas características essenciais na interligação entre o componente e o contexto para que seja possível testar um a partir do outro. A segunda condição é a coesão, que mede a ligação existente entre os elementos dentro de um mesmo subsistema. Em poucas palavras, o componente e o contexto precisam cada um ter uma funcionalidade específica e correlacionada com o do outro.

subsistemas fracamente acoplados e altamente coesos é que caracteriza um sistema bem desenvolvido, segundo as teorias de Engenharia de Software. No caso específico de teste em contexto, os experimentos mostraram que para executá-lo faz-se necessário que o contexto e o componente tenham coesão mediana e, entre si, sejam fortemente acoplados.

Os experimentos práticos feitos para se deduzir tais condições foram a análise da pilha de protocolos do TCP/IP, como previamente mencionado, o cliente e o servidor HTTP e HTTPS via uma servidor Proxy e o cliente e o servidor DHCP via um agente de entrega. A pilha de protocolos do TCP/IP não é passível de se executar teste em contexto porque cada camada foi desenvolvida para ser altamente coesa e acoplada apenas por meio de dados com as camadas adjacentes. Os resultados com um servidor Proxy para HTTPS foram similares: o contexto, o servidor Proxy, e o componente, o servidor HTTPS, são fracamente acoplados.

A análise dos protocolos HTTP via proxy e DHCP via agente de entrega (Capítulo 4) mostrou que é possível executar teste em contexto nestes cenários. Componente e contexto manipulam os mesmos tipos de dados e possuem sinais em comum.

Baseados nos resultados das avaliações práticas foram escolhidas algumas especificações de protocolos para se experimentar a execução de algoritmos para teste em contexto. Os protocolos escolhidos foram o protocolo do Bit Alternante (ABP), que pode ser utilizado para uma transferência confiável de dados sobre um canal susceptível a perdas e a erros de bits; o Protocolo de Controle de Transmissão da Internet (TCP), que permite um fluxo bidirecional e confiável de dados entre duas aplicações remotas na Internet; e a Unidade de Controle do Assinante (SCU), que é um exemplo retirado da literatura.

Foram encontrados três algoritmos principais na literatura para resolver o problema de teste de conformidade em contexto. O primeiro e mais simples deles é a composição das máquinas de estados do componente e do contexto. Há variações desta abordagem que propõem a redução da máquina composta. O importante a observar é que a geração da máquina composta, em si, é um processo custoso. O segundo algoritmo encontrado foi o do teste direto, cujo princípio é obter o máximo da razão entre a porcentagem de transições cobertas e o número total de transições exercitadas nas seqüências de testes. A idéia apresentada é fazer apenas uma busca em profundidade no contexto. O terceiro algoritmo encontrado foi o *hit-or-jump*, cuja idéia principal é fazer caminhadas aleatórias na especificação do contexto em busca de transições que possam exercitar o componente. Desses algoritmos, apenas o algoritmo do teste direto mostrou-se viável de se aplicar. A complexidade dos demais é exponencial.

Uma vez estudados os algoritmos principais existentes na literatura, foi desenvolvido um outro cuja idéia é executar teste em contexto guiado por casos de teste do componente, isto é, mapear seqüências de teste geradas para a máquina de estados do componente em seqüências de teste do contexto.

Os experimentos conduziram à conclusão de que é viável mapear as seqüências de teste do componente ao contexto, conforme o algoritmo proposto, e que ela se dá de maneira mais eficiente quando são feitas a partir de buscas em profundidade no componente. Em termos de complexidade de execução versus cobertura alcançada, o algoritmo do teste direto mostrou-se o mais eficiente.

Todos os algoritmos falharam em exercitar transições do componente que não são ativadas pelo contexto e aquelas relacionadas a tratamento de erros, de temporizadores e de mensagens duplicadas. Isto se deve ao fato de estas transições serem imprevisíveis ao ambiente de onde são trocados os sinais com o contexto, conforme foi discutido no Capítulo 4.

Parece plausível que estes resultados sejam válidos também para uma ampla variedade de campos de conhecimento. No campo da lingüística, em uma comunicação entre três pessoas,

é difícil estabelecer uma comunicação com uma pessoa por meio de outra, a não ser que esta outra seja um tradutor, e neste caso há um forte acoplamento entre a pessoa com quem se deseja comunicar (o componente) e o tradutor (o contexto). No campo da biologia, a compreensão do processo de transdução de sinais de fora para dentro das células tampouco é uma tarefa trivial.

Conclui-se que teste em contexto só é possível de ser exercitado quando há forte acoplamento entre o contexto e o componente e quando este se comporta de maneira previsível ao ambiente através daquele. Nestas condições, a abordagem mais apropriada para resolver o problema é guiar o teste por meio de seqüências de teste geradas no componente. As contribuições específicas deste trabalho foram a investigação das condições sob as quais pode-se efetuar teste de conformidade em contexto, um estudo dos algoritmos existentes na literatura e o desenvolvimento de um novo, a investigação da viabilidade de aplicá-los a protocolos reais utilizados na Internet e a investigação de padrões de desenvolvimento que possam permitir a aplicação de teste de conformidade em contexto. Como trabalho futuro conjecturamos que seja mais viável efetuar teste de conformidade em contextos, isto é, em vez da avaliação do componente em um contexto específico, avaliá-lo em contextos diferentes. Esta abordagem garantiria uma maior certeza de que a implementação de um componente está em conformidade com sua especificação. De maneira similar ao que é utilizado nos algoritmos atuais de classificação de páginas para buscas na Web [Bharat et al., 2001, Pôssas et al., 2005, Calado et al., 2006], a idéia seria classificar a implementação de um determinado componente de acordo com a quantidade de testes em diferentes contextos que ele passou.

Referências Bibliográficas

[Agrawal and Seth, 1988] Agrawal, V. D. and Seth, S. C. (1988). Test generation for VLSI chips. Washington, DC. Computer Society Press.

[Aho and Lee, 1987] Aho, A. V. and Lee, D. (1987). Efficient algorithms for constructing testing sets, covering paths, and minimum flows. In *ATT Bel Labs Tech. Memo. CSTR159*, Murray Hill, NJ.

[Alpern and Schneider, 1985] Alpern, B. and Schneider, F. B. (1985). Defining liveness. *Information Processing Letters*, 21(4):181–185.

[Anderson and Lee, 1990] Anderson, T. and Lee, P. A. (1990). *Fault Tolerance: Principles and Practice*. Springer-Verlag, 2nd edition.

[Anido et al., 2003] Anido, R., Cavalli, A. R., Jr., L. P. L., and Yevtushenko, N. (2003). Test suite minimization for testing in context. *Softw. Test., Verif. Reliab.*, 13(3):141–155.

[Beizer, 1990] Beizer, B. (1990). *Software Testing Techniques*. John Wiley & Sons, Inc., New York, NY, USA.

[Bharat et al., 2001] Bharat, K., Chang, B.-W., Henzinger, M. R., and Ruhl, M. (2001). Who links to whom: Mining linkage between web sites. In *ICDM*, pages 51–58.

[Binder, 1999] Binder, R. V. (1999). *Testing object-oriented systems: models, patterns, and tools*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Bochmann and Petrenko, 1994] Bochmann, G. V. and Petrenko, A. (1994). Protocol testing: review of methods and relevance for software testing. In *ISSTA '94: Proceedings of the 1994 ACM SIGSOFT international symposium on Software testing and analysis*, pages 109–124, New York, NY, USA. ACM Press.

[Boroday et al., 2002] Boroday, S., Petrenko, A., Groz, R., and Quemener, Y.-M. (2002). Test generation for cefsm combining specification and fault coverage. In *TestCom*, pages 355–372.

[Calado et al., 2006] Calado, P., Cristo, M., Gonçalves, M. A., de Moura, E. S., Ribeiro-Neto, B., and Ziviani, N. (2006). Link-based similarity measures for the classification of web documents. *J. Am. Soc. Inf. Sci. Technol.*, 57(2):208–221.

[Cavalli et al., 1999] Cavalli, A. R., Lee, D., Rinderknecht, C., and Zaïdi, F. (1999). Hit-or-jump: An algorithm for embedded testing with applications to in services. In Wu, J., Chanson, S. T., and Gao, Q., editors, *FORTE*, volume 156 of *IFIP Conference Proceedings*, pages 41–56. Kluwer.

[Chauvin, 1975] Chauvin, R. (1975). *L'éthologie: étude biologique du comportement animal*. Presses Universitaires de France, Paris, FR.

[Chow, 1978] Chow, T. S. (1978). Testing software design modeled by finite-state machines. *IEEE Transactions on Software Engineering*, 4(3):178–187.

[Cinderella, 2002] Cinderella (2002). Cinderella SDL.

[Clarke and Wing, 1996] Clarke, E. M. and Wing, J. M. (1996). Formal methods: State of the art and future directions. *ACM Computing Surveys*, 28(4):626–643.

[Consortium, 1994] Consortium, I. S. O. . I. E. (1994). (ISO 9646) information technology - open systems interconnection - conformance testing methodology and framework- part 1: General concepts.

[Constantine et al., 2005] Constantine, L., Fraser, S., Beck, K., Booch, G., Henderson-Sellers, B., McConnell, S., Wirfs-Brock, R., and Yourdon, E. (2005). Echoes?: structured design and modern software practices. In *OOPSLA '05: Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 383–386, New York, NY, USA. ACM Press.

[Cormen et al., 1990] Cormen, T. T., Leiserson, C. E., and Rivest, R. L. (1990). *Introduction to algorithms*. MIT Press, Cambridge, MA, USA.

[Crouch, 1999] Crouch, A. L. (1999). *Design-for-Test for Digital IC's and Embedded Core Systems*. Prentice Hall Modern Semiconductor Design Series. Prentice Hall PTR, New Jersey.

[da Silva Jr. et al., 2000] da Silva Jr., J. L., Sgroi, M., Bernardinis, F. D., Li, S.-F., Sangiovanni-Vincentelli, A. L., and Rabaey, J. M. (2000). Wireless protocols design: challenges and opportunities. In *Proceedings of the Eighth International Workshop on Hardware/Software Codesign, CODES 2000, San Diego, California, USA, 2000*, pages 147–151. ACM.

[Das and Farmer, 1975] Das, P. and Farmer, D. E. (1975). Fault-detection experiments for parallel-decomposable sequential machines. *IEEE Trans. Computers*, 24(11):1104–1109.

[do Rocio Senger de Souza et al., 2001] do Rocio Senger de Souza, S., Maldonado, J. C., and Fabbri, S. C. P. F. (2001). FCCE: Uma família de critérios de teste para validação de sistemas especificados em estelle. In *XV Simpósio Brasileiro de Engenharia de Software (SBES)*, pages 256–271, Rio de Janeiro, BR.

[Dssouli et al., 1999] Dssouli, R., Saleh, K., Aboulhamid, E., En-Nouaary, A., and Bourhfir, C. (1999). Test development for communication protocols: towards automation. *Comput. Networks*, 31(17):1835–1872.

[Eder et al., 1992] Eder, J., Kappel, G., and Schrefl, M. (1992). Coupling and cohesion in object-oriented systems.

[Eppstein, 1990] Eppstein, D. (1990). Reset sequences for monotonic automata. *SIAM J. Comput.*, 19(3):500–510.

[Fabbri and Maldonado, 1994] Fabbri, S. C. and Maldonado, J. C. (1994). Mutation analysis testing for finite state machines. In *Fifth International Symposium on Software Reliability Engineering*, pages 220–229, Monterey, CA. IEEE.

[Fecko et al., 2000] Fecko, M. A., Amer, P. D., Uyar, M. Ü., and Duale, A. Y. (2000). Issues in conformance testing: multiple semicontrollable interfaces. volume 176 of *IFIP Conference Proceedings*, pages 301–313. Kluwer.

[Fernandez et al., 1992] Fernandez, J.-C., Mounier, L., Jard, C., and Jéron, T. (1992). On-the-fly verification of finite transition systems. *Formal Methods in System Design*, 1(2/3):251–273.

[Fielding et al., 1999] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. (1999). Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard). Updated by RFC 2817.

[Friedman and Menon, 1971] Friedman, A. D. and Menon, P. R. (1971). *Fault Detection on Digital Circuits*. Prentice-Hall, Englewood Cliffs, NJ.

[Gamma et al., 1995] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Garey and Johnson, 1979] Garey, M. R. and Johnson, D. S. (1979). *Computer and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, New York.

[Gill, 1961] Gill, A. (1961). State-identification experiments in finite automata. *Information and Control*, 4(2-3):132–154.

[Gill, 1962] Gill, A. (1962). *Introduction to the Theory of Finite-State Machines*. McGraw-Hill, New York.

[Gonenc, 1970] Gonenc, G. (1970). A method for the design of fault detection experiments. *IEEE Transactions on Computers C-19*, pages 551–558.

[Hards, 2004] Hards, B. (2004). A guided tour of Ethereal. *Linux Journal*, 2004(118):7.

[Henniger and Ural, 2000] Henniger, O. and Ural, H. (2000). Test generation based on control and data dependencies within multi-process SDL specifications. In *In Proceedings of the 2nd Workshop on SDL and MSC*, pages 189–202, Col de Porte, Grenoble, France, 2000.

[Hoare, 1985] Hoare, C. A. R. (1985). *Communicating Sequential Processes*. Prentice-Hall.

[Holzmann, 1985] Holzmann, G. (1985). Tracing protocols. *AT&T Technical Journal*, 64(10):2413–2433.

[Holzmann, 1990] Holzmann, G. J. (1990). *Design and Validation of Protocols*. Prentice-Hall, Englewood Cliffs, NJ.

[Hsieh, 1971] Hsieh, E. P. (1971). Checking experiments for sequential machines. In *IEEE Transactions on Computer*, volume C-20, pages 1152–1166.

[Jansch-Pôrto, 1989] Jansch-Pôrto, I. (1989). Fundamentos de tolerância a falhas. *RITA*, 1(3):63–99.

[Jéron, 1991] Jéron, T. (1991). *Contribution à la validation des protocoles: test d'infinitude et vérification à la volée*. PhD thesis, Université de Rennes 1.

[Kohavi, 1990] Kohavi, Z. (1990). *Switching and Finite Automata Theory: Computer Science Series*. McGraw-Hill Higher Education. 2nd ed.

[Kurose and Ross, 2002] Kurose, J. F. and Ross, K. (2002). *Computer Networking: A Top-Down Approach Featuring the Internet*. Addison-Wesley Longman Publishing Co., Inc., Upper Saddle River, NJ, USA.

[Lai, 2002] Lai, R. (2002). A survey of communication protocol testing. *Journal of Systems and Software (JSS)*, 62(1):21–46.

[Laprie, 1985] Laprie, J. C. (1985). Dependable computing and fault tolerance: concepts and terminology. pages 2–11.

[Lee et al., 1996] Lee, D., Sabnani, K. K., Kristol, D. M. ., and Paul, S. (1996). Conformance testing of protocols specified as communicating finite state machines - a guided random walk based approach. In *IEEE Transactions on Communications*, volume 44, pages 631–640.

[Lee et al., 1993] Lee, D., Sabnani, K. K., Kristol, D. M., Paul, S., and Uyar, M. Ü. (1993). Conformance testing of protocols specified as communicating FSMs. In *INFOCOM*, pages 115–127.

[Lee and Yannakakis, 1996] Lee, D. and Yannakakis, M. (1996). Principles and methods of testing finite state machines - A survey. In *Proceedings of the IEEE*, volume 84, pages 1090–1126.

[Lima and Cavalli, 1997] Lima, L. P. and Cavalli, A. R. (1997). A pragmatic approach to generating test sequences for embedded systems. In *IFIP 10th Internation Wokshop on Testing of Communicating SystemsIWPTS*, pages 288–311.

[Lynch and Tuttle, 1989] Lynch, N. A. and Tuttle, M. R. (1989). An introduction to input/output automata. *CWI-Quarterly*, pages 219–246.

[Martins et al., 1999] Martins, E., Sabião, S. B., and Ambrosio, A. M. (1999). Condata: A tool for automating specification-based test case generation for communication systems. *Software Quality Journal*, 8(4):303–319.

[Mateus and Loureiro, 1999] Mateus, G. R. and Loureiro, A. A. F. (1999). *Introdução à Computação Móvel*. Editora da SBC. XI Escola de Computação, Rio de Janeiro, RJ.

[Mealy, 1955] Mealy, G. H. (1955). A method for synthesizing sequential circuits. In *Bell System Technical Journal*, volume 34, pages 1045–1079.

[Menezes et al., 1996] Menezes, A., van Oorschot, P. C., and Vanstone, S. A. (1996). *Handbook of Applied Cryptography*. CRC Press.

[Milner, 1989] Milner, R. (1989). *Communication and Concurrency*. Prentice-Hall International Series in Computer Science.

[Moore, 1956] Moore, E. F. (1956). Gedanken-experiments on sequential machines. In *Automata Studies*, number 34 in Annals of Mathematical Studies, pages 129–153, Princeton, NJ. Princeton University Press.

[Neelakantan and Raghavan, 1995] Neelakantan, B. and Raghavan, S. (1995). Functional software testing. *CONSEG '95 International Conference on S/W Engineering Practices*.

[Pereira et al., 2000] Pereira, C. L., D. C., . J. d. S., Duarte, R. G., Fernandes, A. O., Canaan, L. H., C. J. N., . J. C., and Ambrosio, L. L. (2000). Jade: An embedded systems specification, code generation and optimization tool. In *SBCCI '00: Proceedings of the 13th symposium on Integrated circuits and systems design*, page 263, Washington, DC, USA. IEEE Computer Society.

[Petrenko et al., 1996] Petrenko, A., Yevtushenko, N., von Bochmann, G., and Dssouli, R. (1996). Testing in context: framework and test derivation. *Computer Communications*, 19(14):1236–1249.

[Petri, 1962] Petri, C. A. (1962). Fundamentals of a theory of asynchronous information flow. In *IFIP Congress*, pages 386–390.

[Postel, 1980] Postel, J. (1980). DoD standard Transmission Control Protocol. RFC 761.

[Pôssas et al., 2005] Pôssas, B., Ziviani, N., Wagner Meira, J., and Ribeiro-Neto, B. (2005). Set-based vector model: An efficient approach for correlation-based ranking. *ACM Trans. Inf. Syst.*, 23(4):397–429.

[Sabnani and Dahbura, 1988] Sabnani, K. K. and Dahbura, A. T. (1988). A protocol test generation procedure. *Computer Networks*, 15:285–297.

[Sgroi et al., 2000] Sgroi, M., Lavagno, L., and Sangiovanni-Vincentelli, A. L. (2000). Formal models for embedded system design. *IEEE Design & Test of Computers*, 17(2):14–27.

[Sidhu and Leung, 1989] Sidhu, D. P. and Leung, T.-K. (1989). Formal methods for protocol testing: A detailed study. *IEEE Trans. Software Eng.*, 15(4):413–426.

[Socolofsky and Kale, 1991] Socolofsky, T. and Kale, C. (1991). TCP/IP tutorial. RFC 1180 (Informational).

[Sommerville, 1993] Sommerville, I. (1993). *Software engineering (4th ed.)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Stevens et al., 1999] Stevens, W. P., Myers, G. J., and Constantine, L. L. (1999). Structured design. *IBM Systems Journal*, 38(2/3):231–256.

[Stevens, 1993] Stevens, W. R. (1993). *TCP/IP illustrated (vol. 1): the protocols*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Tanenbaum, 2003] Tanenbaum, A. S. (2003). *Computer Networks, Fourth Edition*. Prentice Hall, Boston, MA, USA.

[Tinbergen, 1989] Tinbergen, N. (1989). *The Study of Instinct*. Oxford University Press, New York, NY, USA.

[Tsunoyama and Naito, 1981] Tsunoyama, M. and Naito, S. (1981). Fault detection for sequential machines by transitions tours. In *Proccedings of the 11th. IEEE Fault Tolerant Computing Symposium*, pages 238–243. Computer Society Press.

[Ural and Williams, 1993] Ural, H. and Williams, A. W. (1993). Test generation by exposing control and data dependencies within system specifications in sdl. In *FORTE*, pages 335–350.

[Verilog, 2002] Verilog (2002). ObjectGEODE simulator.

[von Bochmann and Sunshine, 1980] von Bochmann, G. and Sunshine, C. A. (1980). Formal methods in communication protocol design. In *IEEE Transactions on Communication*, volume 28, pages 624–631.

[Vries et al., 2000] Vries, R. d., Tretmans, J., Belinfante, A., Feenstra, J., Feijs, L., Mauw, S., Goga, N., Heerink, L., and Heer, A. d. (2000). Côte de resyste in PROGRESS. In Foundation, S. T., editor, PROGRESS *2000 – Workshop on Embedded Systems*, pages 141–148, Utrecht, The Netherlands.

[Yevtushenko et al., 1998] Yevtushenko, N., Cavalli, A. R., and Jr., L. P. L. (1998). Test suite minimization for testing in context. In *IWTCS*, pages 127–146.

Apêndice A

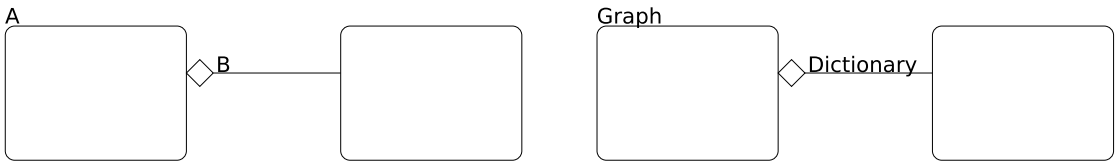
Padrões de desenvolvimento de Orientação a Objetos

O propósito principal desta avaliação é fazer um relacionamento entre padrões de desenvolvimento e sua relação com teste de conformidade em contexto. A ciência da computação está repleta de padrões de desenvolvimento e não há exceção em programação orientada a objetos. Um catálogo deles foi publicado em [Gamma et al., 1995]. Padrões de desenvolvimento não são programas, são escolhas de desenvolvimento amplamente utilizadas para construir programas. Como tal eles formam um tipo de catálogo conceitual que os desenvolvedores são estimulados a utilizar no desenvolvimento de suas aplicações. Eles não são somente úteis para evitar reinventar a roda e economizar tempo, mas também porque provêem uma infraestrutura compreensiva para programadores que necessitam modificar e entender códigos legados. Esta não pretende ser uma apresentação exaustiva de padrões de desenvolvimento orientado a objetos, que já foram bem descritos nas referências citadas.

Há três categorias de padrões de orientação a objetos:

- Padrões criacionais: padrões que podem ser utilizados para criar objetos;
- Padrões estruturais: padrões que podem ser utilizados para combinar objetos e classes visando construir objetos estruturados; e
- Padrões comportamentais: padrões que podem ser utilizados para efetuar uma computação e para controlar fluxos de dados.

Das categorias acima deduz-se que a de padrões estruturais pode ser de interesse para teste de conformidade em contexto. Eles tratam de questões relacionadas a como combinar e estruturar objetos. Por esta razão, vários padrões estruturais fornecem soluções alternativas para



(a) Exemplo simples (b) Grafo implementado como um dicionário

Figura A.1: Diagrama do padrão decorador, procurador ou adaptador

resolver problemas que envolveriam relações de herança entre classes. Os principais padrões nesta categoria são decorador, procurador e adaptador. Todos estes permitem combinar dois (ou mais) componentes, como mostrado na Figuras A.1(a) e A.1(b). Há um componente A respondendo em nome de outro, B. A é o objeto visível que o cliente verá. O papel de A é estender, restringir ou ajudar na utilização de B. Assim, estes padrões são similares a sub-classes, exceto que, enquanto uma sub-classe herda um método de sua classe base, o decorador delega ao decorado quando não possui o método requerido. A vantagem é flexibilidade: pode-se combinar vários componentes em qualquer ordem em tempo de execução sem ter que criar uma hierarquia grande e complexa de sub-classes. Os exemplos mostrados abaixo foram feitos na linguagem de programação Python (a seqüência de caracteres >>> no início da linha indica o *prompt*[1]).

```
>>> class A:
    def __init__ ( self, b ):
        """armazenamento do decorado b
        (b é uma instância da classe B)"""

        self.b = b

def __getattr__ ( self, name ):
        """métodos/atributos que A não conhece
        são delegados a *b*"""

        return getattr ( self.b, name )
```

Durante o seu uso, uma instância da classe A é criada a partir da passagem de uma instância b de B como parâmetro.

```
>>> instituto = "Instituto de Computação - Unicamp"
>>> a = A(instituto)
>>> dir(a)
['__doc__', '__getattr__', '__init__', '__module__', 'b']
>>> a.split()
['Instituto', 'de', 'Computação', '-', 'Unicamp']
>>> 'split' in dir(a)
```

[1] sinal de orientação de que o interpretador da linguagem está pronto para receber comandos.

```
False
>>> 'split' in dir(a.b)
True
>>> len(a)
35
>>> '__len__' in dir(a)
False
>>> '__len__' in dir(a.b)
True
>>>
```

Tudo que a classe A não pode desempenhar é encaminhado para b (estipulando-se que a classe B possa desempenhar).

Padrão	**Descrição**
Decorador	Possibilita adicionar funcionalidades a outro objeto
Procurador	Manipula o acesso a um objeto. Há vários tipos de procuradores: • *de proteção:* para proteger o acesso a um objeto; • *virtual:* para recuperar dados fisicamente somente quando necessário; e • *remote proxy:* para similar um acesso local para um procedimento ativado remotamente.
Adaptador	Ajuda a conectar dois componentes que foram desenvolvidos independentemente e que têm interfaces diferentes.
Composto	Este padrão é freqüentemente utilizado para lidar com composição complexa de estruturas recursivas. A idéia principal é oferecer uma interface uniforme para instâncias de classes diferentes na mesma hierarquia, onde as instâncias são todas componentes do mesmo objeto complexo composto.