Geração de Teste a Partir de Modelos para a Validação de Controles de Acesso

Carlos Diego Nascimento Damasceno

Geração de Teste a Partir de Modelos para a Validação de Controles de Acesso

Carlos Diego Nascimento Damasceno

Orientador: ***Prof. Dr. Adenilso da Silva Simão***

Monografia apresentada ao Instituto de Ciências Matemáticas e de Computação — ICMC — da Universidade de São Paulo, para o Exame de Qualificação, como parte dos requisitos para obtenção do título de Mestre em Ciências de Computação e Matemática Computacional.

USP - São Carlos
Fevereiro/2015

Resumo

Controle de Acesso Baseado em Papeis (do inglês, *Role-Based Access Control* - RBAC) é considerado uma das inovações mais importantes na identificação e gerência de Controle de Acesso (CA). Em sistemas RBAC, o conceito de papel é usado para reduzir a complexidade de tarefas administrativas de segurança, tais como atribuição e revogação de permissões a usuários. Considerando a função crítica desempenhada por sistemas de CA dentro de organizações, a aplicação de métodos formais, tal como Teste Baseado em Modelos, é justificada. Apesar disso, estudos voltados para Teste Baseado em Modelos de Sistemas RBAC são escassos e podem ser considerados incipientes, especialmente em Teste Baseado em Máquinas de Estados Finitos. Nesse contexto, este trabalho propõe-se a investigar técnicas para geração de testes a partir de Máquinas de Estados Finitos para verificação e validação de Sistemas RBAC. Para isso, será realizado um experimento comparando métodos recentes (H, SPY e P) e tradicionais (W e HSI) de geração de casos de teste para Máquinas de Estados Finitos com base nas características, custo e efetividade dos conjuntos de teste gerados. Por meio dos resultados obtidos, espera-se desenvolver um entendimento mais profundo sobre o potencial e limitações das técnicas de Teste de Sistemas RBAC Maseado em Máquinas de Estados Finitos.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo

1

Introdução

1.1 Contextualização

Ataques cibernéticos estão se tornando cada vez mais mais desastrosos, a medida em que a depedência em tecnologia da informação da sociedade atual aumenta. Nesse contexto, diferentes tipos de mecanismos podem ser implementados visando mitigar ameaças e proteger ativos organizacionais (Shabtai et al., 2012). Mecanismos de controle de acesso destacam-se por frequentemente serem usados em combinação com outras soluções para segurança, tais como sistemas de detecção de intrusão e *firewalls*, na tentativa de mediar o acesso de usuários a dados e recursos computacionais (Jaeger e Tidswell, 2000).

Em controle de acesso, o modelo de Controle de Acesso Baseado em Papeis (do inglês, *Role-Based Access Control* - RBAC) é considerado uma das inovações mais importantes na gerência de acesso e identificação (Anderson, 2008). O modelo RBAC utiliza o conceito de agrupamento de privilégios por meio dos papéis de uma organização visando reduzir a complexidade de tarefas de administrativas de segurança, tais como a atribuição e revisão de permissões a usuários (Alturi e Ferraiolo, 2011). Em consequência disso, o RBAC permite mapear mais naturalmente as políticas de segurança na estrutura organizacional de uma instituição.

Em RBAC, um papel consiste de um agente organizacional detentor de um conjunto de responsabilidades. Papéis são definidos e utilizados com o intuito de intermediar a atribuição e revogação de permissões a usuários, facilitando a gerência de segurança (Elliott

e Knight, 2010). Relações hierárquicas entre papéis também podem ser especificadas da mesma forma como alguns cargos ou funções especializam ou generalizam uns aos outros dentro de organizações. Sistemas RBAC também permitem a definição de restrições impedindo a realização de papéis conflitantes através de um conceito denominado Segregação de Funções (do inglês, *Separation of Duties* (SoD). SoD permite especificar políticas que obriguem o envolvimento de múltiplas pessoas para a realização de tarefas. Dessa forma fraudes tornam-se mais difíceis de serem realizadas (Zurko e Simon, 2011). Restrições de cardinalidade também podem ser especificadas a fim de limitar o número de usuários habilitados a usar um dado papel.

Em resposta ao crescente interesse no modelo RBAC, o Instituto Nacional de Padrões e Tecnologia dos Estados Unidos da América (do inglês *National Institute of Standards and Technology* - NIST) propos um documento visando estabelecer um padrão de implementação para o modelo. O padrão ANSI INCITS 359-2004 define um modelo de referência e uma especificação funcional e administrativa para sistemas RBAC que pode ser usado por engenheiros de software interessados em projetar sistemas de controle de acesso RBAC (ANSI, 2004). Atividades voltadas para a verificação e validação desses mecanismos, como é o caso do Teste de Software, também podem ser aplicadas a fim de mitigar e detectar defeitos e vulnerabilidades que possam ser exploradas por atacantes interessados em danificar ou ter acesso privilegiado a dados e recursos computacionais.

Teste de Software é uma atividade do processo de desenvolvimento de software que tem como objetivo principal descobrir situações no qual um sistema apresenta comportamento incompatível com o especificado (Sommerville, 2010). Teste Baseado em Modelos (TBM), por sua vez, é uma variante de teste de software que visa a automatizar a geração de casos de teste a partir de modelos explícitos de especificação formal e critérios sistemáticos de análise a fim de detectar defeitos (Utting et al., 2012). Considerando que a existência de vulnerabilidades em mecanismos de segurança também pode gerar riscos aos ativos de uma organização, o uso de métodos formais, tais como TBM, durante o desenvolvimento de sistemas de controle de acesso pode ser justificado.

1.2 Motivação

Foi constatado que uma crescente a quantidade de estudos publicados nos ultimos anos sobre Teste de Segurança Baseado em Modelos (Damasceno et al., 2014). Nesse contexto, modelos baseados em transição, tais como Máquinas de Estados Finitos (MEF), se apresentam como uma notação frequentemente utilizada. Entretanto, apesar de MEF ser um modelo de especificação que tem sido usado desde a década de 50 para projetar e testar diferentes tipos de sistema, tais como sistemas embarcados, protocolos de comunicação

e circuitos digitais (Broy et al., 2005), estudos envolvendo ambos os tópícos de Teste Baseado em MEF e Sistemas RBAC ainda são escassos e incipientes.

Enquanto na área de Teste Baseado em MEF uma série de trabalhos voltados para a comparação de algoritmos de geração de testes pode ser encontrada (Dorofeeva et al., 2010, 2005b; Endo e Simao, 2013; Simão et al., 2009), revisões de literatura mostraram que a área de Teste de Sistemas de RBAC usando Modelos Baseados em Transição, tais como MEFs, é escassa de estudos experimentais.

No contexto de Teste de Sistemas de Controle de Acesso usando Modelos Baseados em Transição, Masood et al. (2009, 2010b) são um dos poucos autores discutindo a aplicação de Teste Baseado em MEF em Sistemas RBAC. Em seu artigo são propostos uma técnica de teste baseada em MEF Completa e um conjunto de abordagens alternativas para geraçao de testes. As técnicas propostas foram comparadas por meio de um experimento onde MEFs foram geradas a partir das políticas RBAC e testadas usando um único algorítmo de geração de testes para MEF, o método W (Chow, 1978).

Considerando a proposição de novos métodos para teste baseado em MEF é um motivador para comparações envolvendo métodos tradicionais, foi identificado que um estudo envolvendo a aplicação de diferentes métodos de geração de testes para MEF no domínio de sistemas RBAC, assim como feito por Masood et al. (2009), seria pertinente. Um estudo com este enfoque permitiria um entendimento em maior profundidade sobre o potencial do Teste Baseado em MEF para Sistemas RBAC.

1.3 Objetivo

A partir do contexto e da motivação expostos anteriormente, foi definido o seguinte objetivo de pesquisa que será apresentado na seção a seguir.

1.3.1 Objetivo Geral

Da mesma forma como realizado por Endo e Simao (2013), foi estabelecido o objetivo de analisar métodos mais recentes (H, SPY e P) e tradicionais (W e HSI) de geração de sequências para teste de MEFs com o propósito compará-los com relação as características, custo e efetividade dos conjuntos de teste gerados para Teste de Conformidade entre Sistemas e Políticas de Controle de Acesso Baseado em Papel.

A investigação das características, custo e efetividade dos conjuntos de teste gerados pelos métodos citados forneceria uma visão mais detalhada do potencial de cada um dos métodos no contexto de sistemas RBAC. Acredita-se que um comportamento diferente seja obtido considerando diferentes métodos, assim como evidenciado por Endo e Simao (2013), entretanto não existem resultados práticos comprovando tal hipótese no domínio

de sistemas RBAC. Esta proposta de trabalho vem justamente para suprir essa demanda e contribuir com ambas as áreas, de Teste Baseado em MEFs e Controle de Acesso Baseado em Papel.

1.4 Organização do Trabalho

Visando fornecer um embasamento teórico para esse projeto de mestrado, nos próximos cinco capítulos poderão ser encontrados definições que permitirão estabelecer uma conexão entre as áreas de Teste de Software e Controle de Acesso Baseado em Papel.

No Capítulo 2 serão apresentados alguns dos principais conceitos relacionados a Teste de Software. As definições básicas, algumas das principais técnicas e tipos de teste serão discutidos através de exemplos e ilustrações.

No Capítulo 3 será apresentado e discutido o conceito de Teste Baseado em Modelos, uma variante de teste que usa modelos de especificação formal para gerar testes automaticamente. O processo de TBM e uma taxonomia que permite descrever abordagens serão apresentados.

No Capítulo 4 sobre Teste Baseado em Máquinas de Estados Finitos os conceitos de TBM serão estendidos para o domínio de MEFs. Neste capítulo serão apresentados conceitos básicos de MEF, tais como suas propriedades e características, Métodos de Geração de Casos de Teste, Sequências Básicas de Teste e exemplos de ferramentas.

No Capítulo 5 sobre Controle de Acesso serão apresentados conceitos básicos de controle de acesso, os principais modelos e alguns exemplos de softwares de controle de acesso. Devido o foco desse trabalho, uma ênfase maior será dada no Modelo RBAC.

No Capítulo 6 serão discutidos alguns aspectos específicos do tópico que é foco principal desse trabalho: Teste de Controle de Acesso Baseado em Modelos. Nesse capítulo serão apresentados alguns aspectos teóricos que fundamentam a modelagem desse tipo de sistemas e conceitos básicos de Análise e Teste de políticas. Os resultados obtidos por meio de dois estudos sistemáticos também serão apresentados nesse capítulo. A abordagem de Teste de RBAC baseado em máquinas de estados proposta por Masood et al. (2009) também será discutida em detalhes.

Por fim, no Capítulo 7 será apresentado o Plano de Trabalho definido para este projeto de mestrado. Nesse capítulo serão apresentados alguns trabalhos correlatos que auxiliaram na definição do objetivo desse trabalho e na metodologia estabelecida. As atividades e contribuições realizadas pelo aluno até o dado momento, o cronograma de atividades e os resultados esperados para essa pesquisa também serão apresentados e detalhados ao término desse capítulo.

Capítulo

2

Teste de Software

2.1 Considerações Iniciais

A incapacidade humana de executar e comunicar atividades com perfeição oferece inúmeras oportunidades para a ocorrência de problemas durante o processo de desenvolvimento de software (Deutsch, 1981). Além disso, os custos associados a falha de sistemas computacionais têm motivado engenheiros de software a investirem cada vez mais em atividades que ajudem na mitigação e detecção de defeitos, tal como a Verificação e Validação (Pressman, 2001).

Verificação e Validação (V&V) é o conjunto de atividades responsável por medir a qualidade e confiabilidade de softwares, analisar e testar se eles desempenham suas funções corretamente e não operam de modo não intencionado (Wallace e Fujii, 1989). Essas atividades podem ser executadas durante todo o processo de desenvolvimento a fim de garantir o correto funcionamento de um sistema.

Atividades de V&V podem ser divididas em estática, que não requerem a execução ou mesmo existência de um software ou modelo formal, como é o caso das revisões técnicas e *walkthroughs*, e dinâmica, que se baseiam na execução de um programa, como é o caso do teste de software.

Neste capítulo serão expostas definições básicas sobre teste de software (Seção 2.2), processo de teste de software (Seção 2.3) e as principais técnicas de teste (Seção 2.4).

2.2 Definições Básicas

O teste de software possui como objetivo principal descobrir situações no qual um sistema apresenta comportamento incompatível com o especificado (Sommerville, 2010). Por meio do teste, a ocorrência de enganos (do inglês, *Mistake*) durante o processo de desenvolvimento pode ser identificada, reduzindo defeitos de implementação (Delamaro et al., 2007).

Um defeito (do inglês, *Fault*) consiste em um passo, processo ou definição de dados incorreto que pode levar um sistema a um estado inconsistente ou inesperado, gerando um erro (do inglês, *Error*). A ocorrência de um erro, pode se manifestar na forma de falha (do inglês, *Failure*), tal como gerando uma saída incompatível com a esperada para o sistema em teste (do inglês, *System Under Test* - SUT) (IEEE, 1994).

Myers e Sandler (2004) ressaltam que o teste deve objetivar ter uma alta probabilidade de identificar erros, pois isso implica diretamente na sua qualidade. Em consequência disso, o testador deve buscar maximizar a representatividade do subconjunto de casos de teste considerado pois o teste em todos os cenários de operação (teste exaustivo) é impossível ou inviável (Ghezzi et al., 2002). Requisitos de teste podem ser estabelecidos a fim de guiar o processo de teste e se basear em diferentes artefatos de software, tais como código-fonte ou modelos de especificação. Um conjunto de requisitos de teste é denominado *Critério de Cobertura* (Ammann e Offutt, 2008).

Os casos de teste (do inglês, *Test Case*) especificam o conjunto de entradas, condições de execução e resultados esperados para o SUT (IEEE, 1990). O conjunto de todos os caso de teste usados no teste de um SUT é denominado *Conjunto de Teste* (do inglês, *Test Suite*). O conjunto de entrada e de resultados esperados são, respectivamente, subconjuntos finitos do domínio de entrada e de saída do SUT. As condições de execução definem restrições que devem ser satisfeitas para que um caso de teste seja executado. Dados de entrada também são denominados dados de teste (Delamaro et al., 2007).

Ao ser fornecido um dado de entrada, o SUT gera uma saída que deve ser comparada com o resultado esperado. O mecanismo responsável por esta comparação é denominado oráculo de teste (Delamaro et al., 2007). Um oráculo de teste pode ser especificado como uma tabela de valores, algorítmo ou simplesmente do conhecimento do testador (Weyuker, 1982).

2.3 Processo de Teste de Software

Teste de software é uma atividade complexa e que pode envolver um ou mais indivíduos em diferentes tarefas, denominados testadores. Na Figura 2.1 pode ser visto um esquema do processo de teste adaptado de (Ammann e Offutt, 2008).

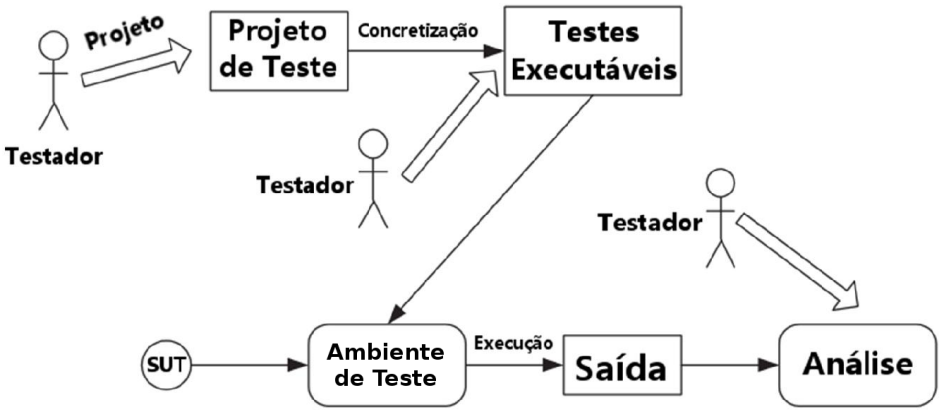


Figura 2.1: Processo de Teste de Software

Primeiramente os casos de teste passam pela etapa de *Projeto de Teste* onde são definidos os requisitos de teste e critérios de cobertura. Estes casos de teste são concretizados na forma de *Testes Executáveis* que serão aplicados no SUT em execução no *Ambiente de Teste*. Ao serem aplicados os dados de teste, o SUT fornece um conjunto de *Saídas* que passam por um processo de *Análise*. Durante a Análise, o oráculo de teste compara a saída obtida com a esperada e decide a aprovação ou falha dos casos de teste.

De modo semelhante, o processo de teste de software também pode ser descrito como um processo de quatro fases (Beizer, 1990; Pressman, 2001): Planejamento, Projeto, Execução e Coleta e Avaliação dos resultados dos testes.

Durante o *Planejamento*, segundo Myers e Sandler (2004), um plano de teste é elaborado contendo os recursos necessários para o teste. Na etapa de *Projeto*, são especificados os dados de entrada, saídas esperadas, critérios de cobertura e requisitos de teste (Myers e Sandler, 2004). Durante a *Execução* são realizados os procedimentos estabelecidos nas etapas anteriores. Na *Análise*, os resultados obtidos são coletados e avaliados por meio do oráculo.

2.3.1 Níveis de Teste de Software

O processo de teste descrito anteriormente pode ser realizado durante todo o processo de desenvolvimento, havendo um tipo de teste para cada atividade do ciclo de vida (Ammann e Offutt, 2008). A Figura 2.2 ilustra as atividade do processo de desenvolvimento de software com seus respectivos testes.

Ainda na etapa inicial do processo de desenvolvimento, durante a análise de requisitos, testes podem ser projetados para avaliar os requisitos capturados. O Teste de Aceitação é que determina se um software completamente desenvolvido e integrado atende de fato

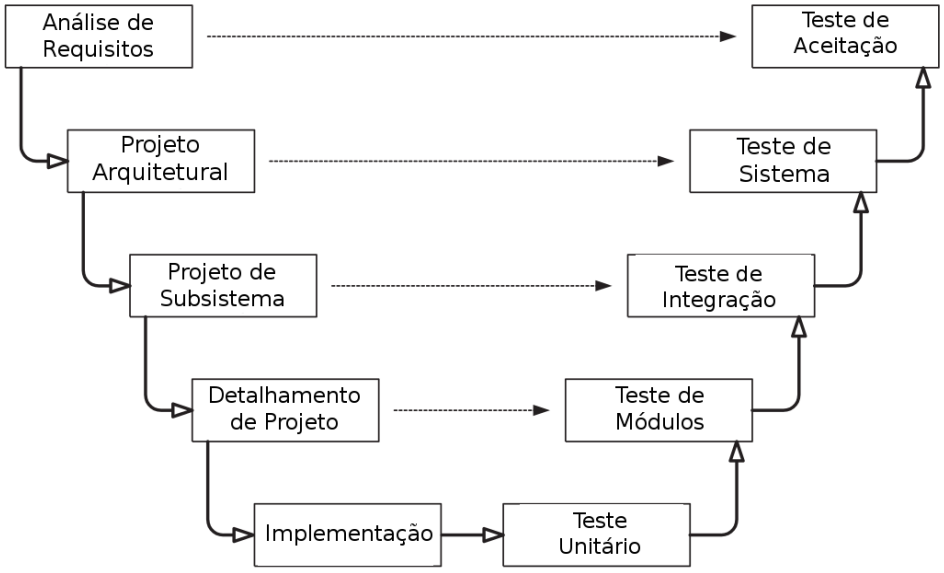


Figura 2.2: Níveis de Teste de Software (Adaptado de (Ammann e Offutt, 2008))

aos requisitos especificados durante a etapa de análise. Ele avalia se o software atende as necessidades especificadas pelo cliente e, por esse motivo, é importante que conte com a participação de clientes, usuários e indivíduos com conhecimento de domínio (Myers e Sandler, 2004).

Durante o projeto arquitetural, a especificação dos componentes e conectores que irão compor o sistema pode ser usada para projetar Testes de Sistema. O Testes de Sistema é aquele projetado a fim de determinar se um sistema montado atende as suas especificações (Pressman, 2001). É importante que os componentes que integram o sistema já tenham sido testados independentemente pois o teste de sistema normalmente procura por problemas de projeto e especificação.

Na fase de projeto de subsistemas, a estrutura e o comportamento dos subsistemas de uma arquitetura pode ser testada por meio do Teste de Integração. O Teste de Integração é projetado para avaliar se as interfaces dos módulos de um subsistema estão corretas e consistentes com a especificação. Nessa etapa deve se assumir o funcionamento correto dos módulos a serem integrados (Pressman, 2001).

A fase de detalhamento de projeto especifica a estrutura e o comportamento dos módulos constituintes de um sistema. Testes de módulos podem ser projetados para avaliar arquivos, pacotes ou classes de um sistema isoladamente. A interação entre módulos também é analisadas com o teste de módulos. Em geral, organizações atribuem a tarefa de teste de módulos aos próprios desenvolvedores dos módulos (Ammann e Offutt, 2008).

Durante a implementação, testes unitários podem ser projetados para analisar softwares no seu nível mais baixo. Em geral, o teste unitário é de responsabilidade dos desen-

volvedores do sistema que podem usar frameworks para implementar as rotinas de teste (Ammann e Offutt, 2008), tal como o JUnit (Husted e Massol, 2004).

Além destes cinco níveis, testes de regressão podem ser aplicados durante a fase de manutenção de um software. O Teste de Regressão é o teste efetuado após modificações serem realizadas em um sistema para assegurar que as suas funções persistirão sem defeitos em versões posteriores (Sommerville, 2010). Outros tipos de teste podem ser encontrados em (Myers e Sandler, 2004)

Cada um dos níveis descritos anteriormente permite a detecção de diferentes tipos de defeitos. Entretanto, como dito anteriormente, para que o número de defeitos e as suas chances de detecção sejam maximizadas, a construção dos casos de teste deve ser feita segundo critérios de seleção bem estabelecidos que normalmente estão diretamente relacionados a técnica de teste.

2.4 Técnicas de Teste

Técnicas de teste podem ser divididas em quatro tipos: Teste Funcional, Teste Estrutural, Teste Baseado em Defeitos e Teste Baseado em Modelos. As três primeiras técnicas de teste serão brevemente descritas nas seções a seguir. Por ser o foco principal desse trabalho de mestrado, Teste Baseado em Modelos será discutido em maiores detalhes nos Capítulos 3, 4 e 6.

2.4.1 Teste Funcional

Teste funcional é uma técnica usada para projetar casos de teste caixa preta, ou seja, quando não se tem conhecimento do funcionamento interno ou código-fonte do programa. Neste tipo de teste são considerados somente os casos de teste contendo as entradas e as saídas do SUT. A partir deste par entrada/saída, o testador verifica se o programa está em conformidade com sua especificação (Myers e Sandler, 2004).

Em princípio, o teste funcional pode detectar todos os defeitos de um programa quando forem consideradas todas as entradas e combinações possíveis, o que denomina-se teste exaustivo (Delamaro et al., 2007). Entretanto, a usual dimensão significativa ou infinita do domínio de entrada dos programas torna esse teste inviável. Para tratar este problema, critérios de cobertura podem ser usados para aumentar o rigor do processo de teste. Os critérios de cobertura *Particionamento em Classe de Equivalência*, *Análise de Valor Limite* e *Teste Funcional Sistemático* serão discutidos a seguir.

Particionamento em Classe de Equivalência

Particionamento em Classe de Equivalência visa fragmentar o domínio de entrada em partes denominadas classes de equivalência. Ao serem definidas classes de equivalência, podemos assumir com alguma segurança que qualquer elemento de uma classe pode ser dito representante de todos os demais membros dessa mesma classe. Isso permite a redução do domínio de entrada a um tamanho passível de ser tratado por completo (Myers e Sandler, 2004).

Além do domínio de entrada, o domínio de saída também pode ser dividido em classes de equivalência. Um meio de identificar estas classes é buscando na especificação do programa termos como intervalo, conjunto ou outras palavras que indiquem que dados são processados do mesmo modo (Delamaro et al., 2007). A seguir será apresentada a especificação do programa "Desconto de Passagem" para fornecimento de descontos em compras de passagens.

> Clientes cadastrados como estudante têm direito a 50% de desconto no preço da passagem. Clientes com idade inferior a 4 anos ou igual ou superior a 60 anos podem adquirir gratuitamente uma passagem. Caso um cliente esteja apto a mais de um desconto, o maior desconto prevalece.

Considerando o exemplo citado, podemos definir dois parâmetros de entrada: *Estudante* e *Idade*. Para *Estudante* podemos definir os valores *Sim* ou *Não* e para *Idade* um valor inteiro positivo. Considerando os valores de saída, temos três alternativas: *0% de desconto*, *50% de desconto* e *100% de desconto*. A partir disso podemos identificar as seguintes classes de equivalência válidas e inválidas, apresentadas na Tabela 2.1.

Tabela 2.1: Classe de Equivalência da Especificação "Desconto de Passagem"

Variável	**Classe Válida** (Com Desconto)	**Classe Inválida** (Sem Desconto)
Estudante	Sim	Não
Idade	(Idade<4) || (Idade >=60)	(Idade<1) || ((4<=Idade) && (Idade<60))
Desconto	50% || 100%	0%

Identificadas as classes de equivalência, casos de teste podem ser definidos usando elementos escolhidos arbitrariamente de cada classe. Na Tabela 2.2 será apresentado um conjunto de casos de teste definido usando as classes de equivalência identificadas no programa "Desconto de Passagem".

Análise de Valor Limite

O critério de análise de valor limite sugere que valores que exploram condicões limites têm maior probabilidade de encontrar defeitos (Myers e Sandler, 2004). Condições limite

Tabela 2.2: Casos de Teste definidos usando Classes de Equivalência

Variáveis de Entrada		**Saída Esperada**
Estudante	**Idade**	**Desconto**
Sim	15	50%
Não	15	0%
Não	2	100%
Não	62	100%
Não	0	0%

correspondem aos valores que se encontram exatamente sobre ou imediatamente acima ou abaixo dos limitantes de classes de equivalência (Delamaro et al., 2007).

A técnica de análise de valor limite é usada em conjunto com o particionamento em classes de equivalência entretanto, ao invés de selecionar valores aleatoriamente, o testador deve voltar a atenção para valores limite. A seguir, na Tabela 2.3 serão ilustrados casos de teste gerados com base na especificação do programa "Desconto de Passagem" usando a técnica de análise de valor limite.

Tabela 2.3: Casos de Teste definidos usando Análise de Valor Limite

Variáveis de Entrada		**Saída Esperada**
Estudante	**Idade**	**Desconto**
Não	0	0%
Não	1	100%
Não	3	100%
Não	4	0%
Sim	18	50%
Não	19	0%
Não	59	0%
Sim	60	100%

Teste Funcional Sistemático

O critério de teste funcional sistemático combina os critérios de particionamento de equivalência e análise de valor limite. Ele requer que, uma vez que os domínios de entrada tenham sido definidos e particionados em classes de equivalência, o limite de cada partição seja testada assim como ao menos dois casos de teste de cada partição (Linkman et al., 2003). O critério também fornece um conjunto de diretrizes para testar domínios de entrada e saída usando intervalos, valores ilegais, numéricos, reais, *arrays* e *strings* (Delamaro et al., 2007). A seguir, serão ilustrados na Tabela 2.4 um conjunto de casos de teste gerado usando a especificação do programa "Desconto de Passagem" e a técnica de Teste Funcional Sistemático.

Tabela 2.4: Casos de Teste definidos usando Teste Funcional Sistemático

Variáveis de Entrada		**Saída Esperada**
Estudante	**Idade**	**Desconto**
Não	0	0%
Não	1	100%
Não	2	100%
Não	3	100%
Não	4	0%
Sim	4	50%
Sim	18	50%
Não	25	0%
Sim	59	50%
Não	59	0%
Sim	60	100%
Sim	61	100%
Não	65	100%
Não	70	100%

2.4.2 Teste Estrutural

A técnica de teste estrutural define os requisitos de teste a partir da implementação do programa, requerendo a execução de partes ou componentes elementares do seu código (Delamaro et al., 2007). Para isso, o programa pode ser representado na forma de um Grafo de Fluxo de Controle (GFC). Em um GFC, os comandos de um programa são representados na forma de nós e os fluxos de controle (decisões) como arestas. Na Figura 2.3 pode ser visto um exemplo de gráfico de fluxo de controle.

A seguir, serão discutidos três dos principais critérios de teste estrutural: Critérios baseados em complexidade, Critérios baseados em fluxo de controle e Critérios baseados em fluxo de dados.

Critérios Baseados na Complexidade

Critérios baseados na complexidade usam a complexidade de um programa para derivar requisitos de teste. O critério de McCabe é um exemplo de critério de cobertura que utiliza a complexidade ciclomática do GFC para gerar testes. Complexidade ciclomática é uma métrica de software que fornece uma medida quantitativa sobre a complexidade lógica de um programa (McCabe, 1976). Segundo Pressman (2001), a complexidade ciclomática também pode ser usada como medida quantitativa descrevendo a dificuldade para testar um programa ou a confiabilidade de um conjunto de testes.

No contexto de teste de software, o valor da complexidade é usado para estabelecer o número de caminhos linearmente independentes em um programa. Um caminho linear-

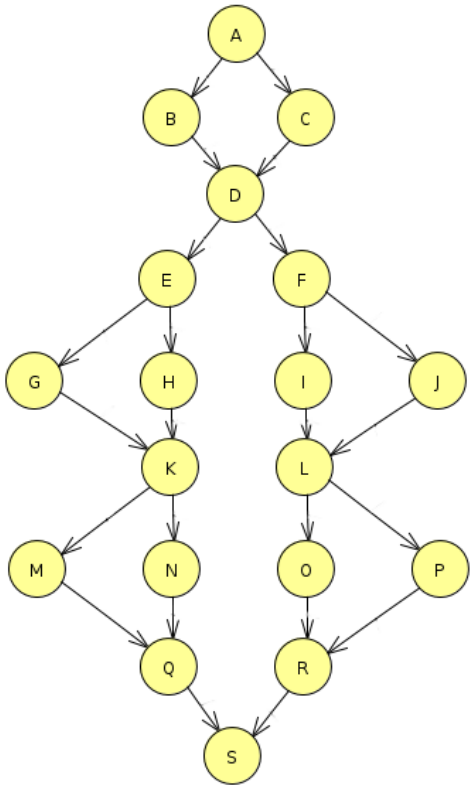


Figura 2.3: Exemplo de Gráfico de Fluxo de Controle

mente independente consiste de qualquer caminho no código do programa que introduza um novo conjunto de transições ou condição. Para calcular a complexidade ciclomática de um GFC usamos a Equação 2.1.

$$\mathcal{C} = \text{total\_arcos} - \text{total\_nós} + 2 \tag{2.1}$$

A partir disso, podemos definir a complexidade ciclomática do gráfico de fluxo de controle da Figura 2.3 como $\mathcal{C} = 24 - 19 + 2 = 7$ e os seus respectivos caminhos como $\{ABDEGKMQS, ACDEGKMQS, ABDFILORS, ABDEHKMQS, ABDEGKNQS, ACDFJLORS, ACDFILPRS\}$.

Critérios Baseados em Fluxo de Controle

Critérios baseados em fluxo de controle utilizam características do controle de execução do programa (i.e., comandos ou desvios) como requisitos de teste. Comandos e desvios são descritos nos GFCs como, respectivamente, nós e arestas que devem ser atravessados (cobertos) por casos de teste segundo os critérios estabelecidos. Os três principais critérios baseados em fluxo de controle são (Myers e Sandler, 2004; Pressman, 2001):

- **Todos-Nós:** Define que cada um dos nós do GFC seja coberto pelo menos uma vez.

- **Todas-Arestas:** Define que cada aresta do GFC seja exercitada pelo menos uma vez.

- **Todos-Caminhos:** Define que todos os caminhos possíveis no GFC sejam executados.

É importante ressaltar que em certos casos pode haver nós/arestas impossíveis de serem executadas. Consequentemente, não haverá garantia de uma cobertura de 100% dos nós, arestas ou caminhos do GFC (Ammann e Offutt, 2008). O total de caminhos também pode ser um número muito grande ou até mesmo infinito, tal como em programas com *loops*. Este foi um dos fatores que motivou a criação dos Critérios Baseados em Fluxo de Dados.

Critérios Baseados em Fluxo de Dados

Critérios baseados em fluxo de dados analisam o fluxo de valores de variáveis ao longo do código para derivar requisitos de teste. Estes critérios consideram a interação entre uma definição e o uso subsequente de variáveis (def-uso, ou simplesmente *du*) para assegurar que elas estão sendo criadas e usadas corretamente (Rapps e Weyuker, 1982). Para isso, Rapps e Weyuker (1982) extenderam o conceito de grafo GFC com informações referentes à definição e uso de variáveis, o que foi denominado Grafo Def-Uso (do inglês, *def-use graph*).

Uma definição (*def*) consiste de um ponto no programa onde um valor é atribuído a uma variável (e.g., $x = 1$). Um uso (*use*) é o local em um programa onde uma variável é referenciada (e.g., $x = x + 1$). Quando uma variável é usada em uma condição (e.g., $x \geq 1$) temos um uso predicativo (p-uso), caso contrário, temos um uso computacional (c-uso) (e.g., $x = a + 2$). Os principais critérios são:

- **Todas-Definições (*all-def*):** Requer que cada definição e uso (c-uso ou p-uso) de uma variável seja exercitadas pelo menos uma vez.

- **Todos-Usos (*all-uses*):** Requer que todas as associações entre definições e usos (c-uso ou p-uso) de variáveis sejam exercitadas por pelo menos um caminho livre de definição da variável testada.

- **Todos-Du-Caminhos (*all-du-paths*):** Requer que toda a associação entre a definição de uma variável e seus usos subsequentes seja exercitada por todos os caminhos livres de definição e livres de laço que cubram essa associação.

2.4.3 Teste Baseado em Defeitos

Teste baseado em defeitos (do inglês, *fault based testing*) utiliza-se de falhas frequentemente existentes em programas e que tenhamos interesse de detectá-las (Demillo et al., 1978). Os principais critérios deste tipo são Teste de Mutação e Semeadura de Erros, descritos a seguir.

Teste de Mutação

Teste de Mutação, também denominado de Análise de Mutantes, utiliza defeitos típicos do processo de implementação de software para derivar requisitos de teste. Nessa técnica são geradas versões alternativas (mutantes) do programa original (P) por meio de operadores de mutação. Esses mutantes (P') formam uma vizinhança de programas, $\phi(P)$, que difere do programa original P devido pequenas alterações. Após a geração dos mutantes, o testador deve projetar um conjunto de testes (T) que consiga evidenciar mutantes com comportamento diferente do original. Usando essa estratégia, o testador pode calcular uma métrica de adequação para o conjunto de testes T e para o programa em teste P, denominada "escore de mutação" ($ms(P,T)$). Essa métrica é calculada da seguinte maneira:

$$ms(P,T) = \frac{DM(P,T)}{M(P)-EM(P)} \tag{2.2}$$

onde:

DM(**P**, **T**) consiste do número de mutantes identificados como diferentes de P, considerando o caso de teste T, denominados mutantes mortos.

M(**P**) consiste do total de mutantes gerados a partir de P.

EM(**P**) consiste do número de mutantes equivalentes a P.

Observe que apenas $DM(P,T)$ depende dos casos de teste. Os demais valores são obtidos iterativamente pelo testador pois inicialmente não se tem conhecimento do total de mutantes nem de quais apresentam comportamento equivalente a P.

O teste de mutação defende sua efetividade na identificação de casos de teste adequados assumindo duas hipóteses: Hipótese do Programador Competente e Efeito do Acoplamento.

A Hipótese do Programador Competente, proposta por Demillo et al. (1978), define que o programador é um ser competente e tem a capacidade de desenvolver programas próximos do correto. Em consequência disso, apesar dele poder cometer falhas, pequenas

modificações sintáticas podem corrigi-las. Por esta razão, modificações sintáticas simples podem ser usadas para representar os erros que estes "programadores competentes" venham a cometer.

Diferentemente da Hipótese do Programador Competente, a Hipótese do Efeito do Acoplamento (Demillo et al., 1978) define que casos de teste que consigam distinguir programas mutantes gerados por meio de erros simples, ou seja, uma única modificação, são suficientemente sensíveis ao ponto de terem a capacidade de distinguir erros mais complexos.

É importante ressaltar que a vizinhança $\phi(P)$ deve ser suficientemente abrangente e tenha uma cardinalidade pequena. Essas características garantem a obtenção de casos de teste capazes de identificar uma maior quantidade de erros e uma quantidade de casos de teste tratável (Delamaro et al., 2007). A cardinalidade da vizinhança é um ponto crítico pois, assim como provado por Budd e Angluin (1982), identificar a equivalência entre um programa e seus mutantes é um problema indecidível, denominado Problema de Detecção de Mutantes Equivalentes (Budd e Angluin, 1982). Em consequência disso, essa tarefa deve ser realizada manualmente.

Apesar de ser considerada uma técnica cara, principalmente devido o Problema de Detecção de Mutantes Equivalentes, o Teste de Mutação tem se mostrado um dos mais efetivos para revelar defeitos. Por esta razão, diversos trabalhos de cunho experimental têm utilizado teste de mutação para verificar a efetividade de novos critérios de teste (Andrews et al., 2006; Do e Rothermel, 2006). Jia e Harman (2011) também fornecem evidências apontando um crescimento no interesse pela área e do seu grau de maturidade.

A seguir, no Algorítmo 2.1, temos um exemplo simplificado de teste de mutação do programa "Desconto de Passagem", sendo *idade* uma variável do tipo *int* e *estudante* do tipo *boolean*.

```
1 if((idade >= 60) || (idade>=1 && idade<4 )){
2   System.out.println("100% de desconto");
3 }else if (estudante==true){
4   System.out.println("50% de desconto");
5 }else{
6   System.out.println("0% de desconto");
7 }
```

Algorítmo 2.1: Desconto de Passagem

Se considerarmos um operador de mutação que altere o operador lógico ">=" por ">" e aplicarmos no programa exemplo, uma possibilidade de mutante P' gerado teria a seguinte forma apresentada no Algorítmo 2.2:

```
1 if((idade > 60) || (idade>=1 && idade<4 )){
2   System.out.println("100% de desconto");
```

```
}else if (estudante==true){
  System.out.println("50% de desconto");
}else{
  System.out.println("0% de desconto");
}
```

Algorítmo 2.2: Desconto de Passagem - Mutante

Nesse caso, o primeiro predicado **(idade >= 60)** foi alterado para **(idade > 60)** e um caso de teste considerando um cliente com idade igual a 60 seria capaz de evidenciar a diferença P de P'. Por outro lado, se considerarmos um operador de mutação que altere predicados tal como no Algorítmo 2.3, mudando **(estudante==true)** para **(estudante!=false)**, não haveria caso de teste capaz de diferenciar P de P', logo seriam equivalentes.

```
if((idade >= 60) || (idade>=1 && idade<4 )){
  System.out.println("100% de desconto");
}else if (estudante!=false){
  System.out.println("50% de desconto");
}else{
  System.out.println("0% de desconto");
}
```

Algorítmo 2.3: Desconto de Passagem - Mutante Equivalente

Semeadura de Erros

Semeadura de erros é o processo de intencionalmente adicionar defeitos conhecidos em um programa de computador com o propósito de monitorar a taxa de detecção e remoção, e estimar o número de defeitos remanescentes no programa (IEEE, 1990). O problema com essa abordagem é que os defeitos artificiais podem mascarar no sistema defeitos reais e, para tornar os resultados mais confiáveis, sugere-se considerar programas que suportem a injeção de pelo menos 10.000 defeitos. Além disso, esta técnica assume que os defeitos possuem uma distrubuição uniforme ao longo do programa, o que nem sempre é o caso (Vincenzi et al., 2010).

2.5 Considerações Finais

Neste capítulo foram introduzidos os conceitos básicos de teste de software. A terminologia específica da área de teste também foi apresentada de modo que palavras e conceitos possam ser corretamente assimilados pelo leitor. Um processo de teste foi apresentado a fim de ilustrar as atividades e o modo como elas são conduzidas. Os níveis onde o

teste de software pode ser aplicado durante um processo de desenvolvimento também foram apresentadas assim como a relação de cada um desses níveis de teste com as etapas do ciclo de vida. Um conjunto de técnicas e critérios de teste foi apresentado a fim de fornecer uma visão geral de como o teste pode ser aplicado e avaliado. As técnicas de teste funcional, estrutural e baseado em defeitos foram detalhadas. As informações apresentadas nesse capítulo são fundamentais para o entendimento dos demais capítulos, principalmente aqueles relacionados a variantes de teste, como é o caso do TBM e do Teste Baseado em MEFs.

Capítulo

3

Teste Baseado em Modelos

3.1 Considerações Iniciais

Da mesma forma que ocorre durante o projeto de um sistema, o testador deve buscar meios de expressar casos de teste com o mínimo de ambiguidade e máximo de detalhameto. Para isso, ele deve idealmente utilizar notações que descrevam os objetivos do teste e características do SUT de modo mais completo e simples possível. Neste caso, o uso de notações explícitas, como UML e máquinas de estados finitos, podem ser de grande ajuda na descrição e até mesmo geração e execução de artefatos de teste. Teste Baseado em Modelos é o processo de teste que utiliza modelos de especificação de sistemas para gerar artefatos de teste de modo automático.

Neste capítulo serão discutidos conceitos básicos de teste baseado em modelos, etapas do processo de teste baseado em modelos e uma taxonomia, proposta por (Utting et al., 2012), que permite descrever metodologias de teste que usem modelos segundo alguns parâmetros.

3.2 Teste Baseado em Modelos

Teste Baseado em Modelos (TBM) é uma variante de teste que visa a automatizar a geração de casos de teste a partir de modelos explícitos de especificação formal e critérios sistemáticos de análise (Utting e Legeard, 2007). TBM busca solucionar problemas típicos

do processo tradicional de teste que usualmente tende a ser manual, pouco sistemático, mal documentado, custoso e tedioso (Utting et al., 2012).

Segundo Utting e Legeard (2007), TBM pode entendido segundo quatro definições com diferentes focos, porém complementares:

1. Geração de dados de entrada para casos de teste a partir de descrições formais do modelo de domínio do SUT;
2. Geração de casos de teste a partir de modelos do ambiente do sistema em teste;
3. Geração de casos de teste concretos a partir de casos de teste abstratos; e
4. Geração de casos de teste com oráculos a partir de modelos comportamentais.

A combinação destas definições torna o TBM bem mais sofisticado e flexível quando comparado com as demais definições isoladamente (Utting e Legeard, 2007). Por esta razão, TBM será definido como o processo de geração de casos de teste concretos com oráculos a partir de modelos comportamentais.

3.3 Etapas do Processo de Teste Baseado em Modelos

O processo de geração de casos de teste pode ser dividida em cinco etapas (Figura 3.1): Modelagem, Geração, Concretização, Execução e Análise. Essas etapas serão discutidas nas seções a seguir.

3.3.1 Modelagem de Sistemas em Teste

Durante a modelagem, o comportamento do SUT é descrito a partir da sua especificação de requisitos usando notação formal. Esse modelo é normalmente denominado *Modelo de Teste* pois o seu nível de detalhamento e foco são definidos com base nos objetivos do Plano de Teste. Modelos de projeto também podem ser reutilizados para este fim. Entretanto, essa prática de reuso não é muito aconselhada pois erros de projeto podem se propagar para a etapa de teste. Modelos de teste também podem ser validados com base na sua completude e consistência. Dessa forma, é interessante que eles se apresentem de modo mais simples, quando comparado com o SUT e com o modelo de projeto.

3.3.2 Geração de Casos de Teste

Durante a fase de geração de casos de teste, critérios de seleção são utilizados para guiar a geração automática de "bons" conjuntos de testes. Um conjunto de teste é dito "bom"

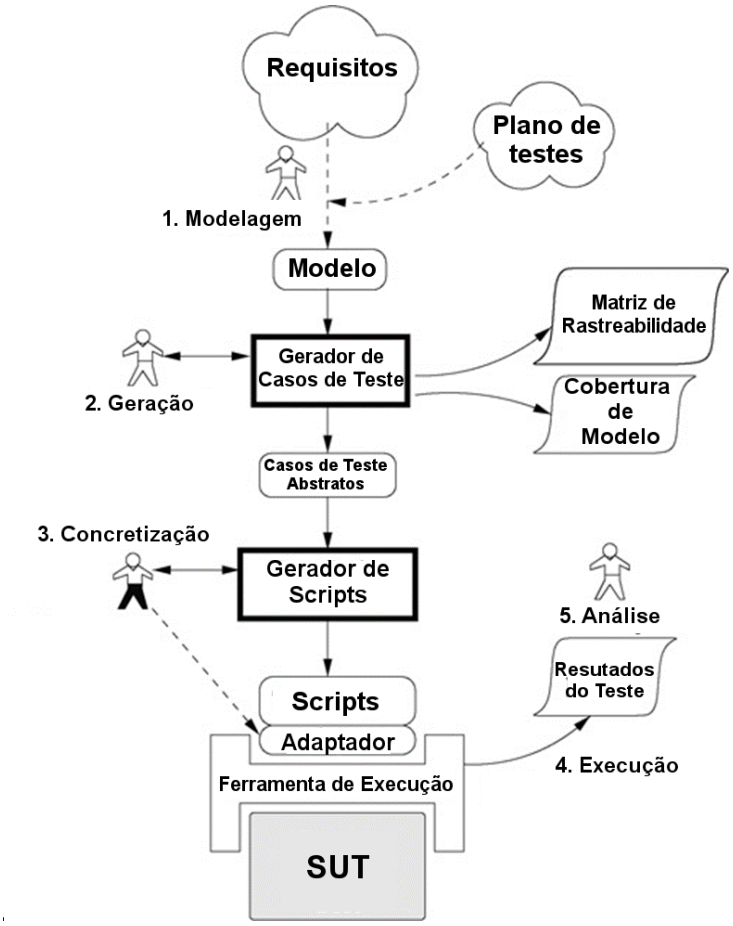


Figura 3.1: Processo do Teste Baseado em Modelos

quando ele atende aos requisitos estabelecidos no plano de teste. Critérios de seleção podem utilizar os requisitos funcionais do SUT, estrutura do modelo de testes ou domínio de dados de entrada/saída para avaliar o grau de cobertura dos conjuntos de teste. Casos de teste também podem ser mapeados aos requisitos do SUT usando matrizes de rastreabilidade. Vale ressaltar que nessa etapa os casos de teste são ditos abstratos pois não estão num formato executável. Para serem executados, os casos de teste devem ser convertidos para um formato apropriado. A etapa responsável pela conversão de casos de teste abstratos em executáveis é denominada Concretização.

3.3.3 Concretização de Casos de Teste

Durante a concretização, os casos de teste abstratos são mapeados para um formato executável. Segundo Utting et al. (2012) existem três principais técnicas de concretização e a mais indicada depende do tipo de modelo, comportamento do SUT e do tipo de teste: Adaptação, Transformação e Híbrida (Figura 3.2).

- **Adaptação:** um adaptador é utilizado para realizar a ponte entre os casos de teste abstratos e o SUT.

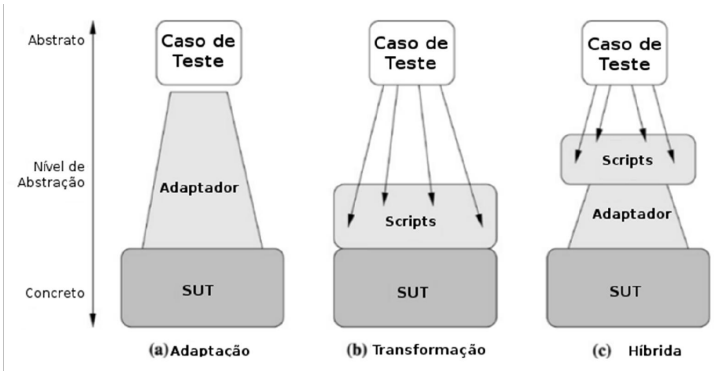


Figura 3.2: Técnicas de Concretização de Teste

- **Transformação:** os casos de teste abstratos são convertidos para um formato executável.
- **Híbrida:** os casos de teste são convertidos para *scritps* que são processados por um adaptador.

3.3.4 Execução de Casos de Teste

A etapa de execução do teste diz respeito ao tempo relativo entre a geração e execução dos casos de teste que pode ser efetuada de modo *online* ou *offline*. Em uma execução de casos de teste *offline*, a geração e execução de casos de teste ocorre em etapas separadas e sequencialmente. Na execução *online*, os algorítmos de geração de testes enviam entradas e devem reagir as saídas e estados que o SUT apresentar.

A execução de casos de teste também pode ser descrita como manual ou automática. No teste manual, casos de teste são aplicados por um indivíduo interagindo diretamente com o SUT. O teste automático, por sua vez, consiste da aplicação de casos de teste com pouca ou nenhuma intervenção humana.

3.3.5 Análise de Casos de Teste

Durante a análise, os resultados obtidos com a execução dos casos de teste são avaliados e utilizados para tomar medidas corretivas. Em TBM, o processo de análise de casos de teste é realizado de modo semelhante ao do processo de análise no teste tradicional. Um oráculo de teste é utilizado para fornecer o veredito de aprovação ou falha para os casos de teste e, para cada falha identificada, o defeito que a causou deve ser determinado. Em TBM, falhas também podem ser identificadas devido modelos de teste inconsistentes, problemas no adaptador ou em casos de teste, e não somente devido defeitos no SUT. Por este motivo, sugere-se que o modelo de teste seja validado antes de ser encaminhado para a etapa de geração e execução de testes.

3.4 Taxonomia do Teste Baseado em Modelos

Como visto na seção anterior, um processo de teste baseado em modelos pode ser dividido em cinco etapas onde diversos artefatos são utilizados e gerados de modo automático e manual. Por sua vez, artefatos gerados durante um processo de TBM e a própria abordagem podem ser categorizados segundo uma taxonomia proposta por Utting et al. (2012).

A taxonomia do TBM proposta por Utting et al. (2012) permite descrever abordagens de teste baseado em modelos e seus artefatos segundo sete dimensões: (1) Assunto, (2) Independência, (3) Características e (4) Paradigma do Modelo de Teste; (5) Critério de Seleção e (6) Tecnologia de Geração de Testes; e (7) Execução de Testes. Cada uma dessas sete dimensões pode ser visualizada na Figura 3.3 e será discutida nos parágrafos a seguir. Os elementos nas folhas da taxonomia separados por barras ("A/B") são mutuamente exclusivos, enquanto que aqueles conectados por linhas curvas definem características que podem simultaneamente fazer parte de uma mesma abordagem de TBM.

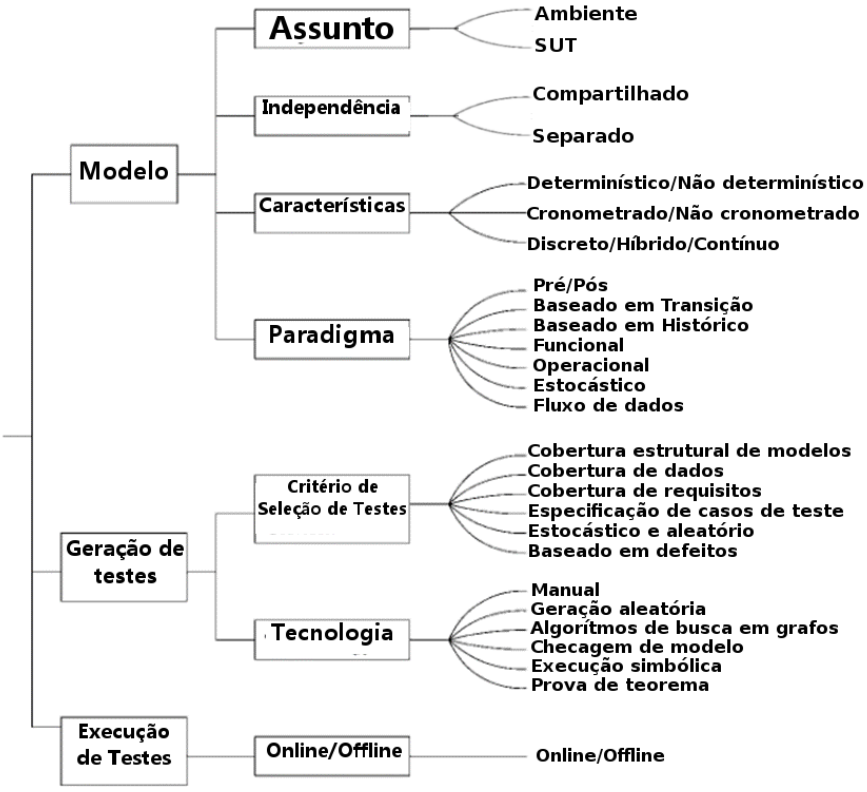


Figura 3.3: Taxonomia do Teste Baseado em Modelos

Assunto é a dimensão que diz respeito aquilo que se encontra descrito no modelo de teste. Um modelo de teste pode descrever o comportamento de um SUT, o ambiente no qual os SUT estará inserido ou até mesmo ambos. Em abordagens de TBM onde modelos

de ambiente de SUT são usados, testadores podem especificar o perfil de uso de um SUT e voltar a geração de testes para este perfil estabelecido.

Independência é a dimensão que reflete a origem do modelo usado na atividade de teste. Como dito anteriormente, um modelo de teste pode ter sido criado especialmente para a atividade de TBM ou ter reutilizado modelos de projeto pré-existentes. No primeiro caso, diz-se que o modelo tem independência separada, enquanto que no segundo tem independência compartilhada.

Característica é a dimensão que diz respeito a natureza comportamental e propriedades do SUT incorporadas no modelo de teste. As informações agregadas no modelo geralmente refletem características do sistema em teste, tais como restrição de tempo, não determinismo e interação baseada em eventos discretos ou contínuos. Modelos de teste de SUTs, tais como aplicações paralelas ou sistemas de tempo real, devem conter informações que expressem de modo consistente sua natureza.

Paradigma é o parâmetro que descreve a notação usada para especificar o comportamento do SUT. Utting et al. (2012) agrupa as notações usadas no teste baseado em modelos em sete categorias, listadas a seguir:

- **Notação Pré-Pós:** Permitem modelar um sistema como uma coleção de variáveis representando retratos do estado interno do SUT junto das suas operações. Z, B e OCL são alguns exemplos de notação dessa categoria.

- **Notação Baseada em Transições:** Dão ênfase na especificação das transições existentes entre os estados que um sistema pode assumir. Máquinas de Estados Finitos e *statecharts* são exemplos de notação dessa categoria.

- **Notação Baseada em Histórico:** Descrevem o *trace* do comportamento de um SUT. Lógicas temporais são um exemplo de notação para especificar o histórico do comportamento de SUTs.

- **Notação Funcional:** Descrevem um SUT como uma coleção de funções matemáticas usando lógica de primeira ordem ou de ordem superior, tais como HOL.

- **Notação Operacional:** Descreve o SUT como um conjunto de processos executáveis em paralelo. Frequentemente usada para especificar sistemas distribuiídos e protocolos de comunicação. Exemplos de notação são Redes de Petri e VHDL.

- **Notação Estocástica:** Frequentemente usada para descrever o ambiente do SUT com modelos probabilísticos representando eventos e entradas. Modelos de markov são um exemplo de notação usada para este fim.

- **Notação Fluxo de Dados:** Descreve o fluxo de dados, ao invés do fluxo de controle de um SUT. O modelo de diagrama de blocos do Matlab Simulink é um exemplo de notação deste tipo.

Na prática, diferentes notações podem ser usadas para especificar um SUT segundo diferentes visões, tais como o comportamento dinâmico e regras de negócio. A Unified Modeling Language (UML), por exemplo, combina uma série de notações, tais como máquinas de estados e OCL.

Critério de Seleção de Teste é a dimensão que define a abordagem usada para controlar a geração de testes. Essa dimensão define o conjunto de requisitos que os conjuntos de teste a serem gerados devem satisfazer. Critérios de seleção usados no teste de software tradicional, tais como cobertura estrutural, também podem ser usados em TBM. Exemplos de critérios de seleção são listados a seguir.

- **Critério de Cobertura Estrutural:** Este tipo de critério explora a estrutura do modelo do SUT. Em geral, a própria notação sugere meios de mensurar o grau de cobertura que um teste aplica sob um modelo, tais como transições e estados. Em MEFs, exemplos de critérios de cobertura estrutural são Método W, Wp, HSI e SPY (Endo e Simao, 2013).

- **Critério de Cobertura de Dados:** Critérios dessa categoria têm o objetivo de criteriosamente selecionar subconjuntos de casos de teste em grandes espaços de busca. Essencialmente, a estratégia desses critérios consiste de dividir um espaço de busca em classes de equivalência, no que tange sua capacidade de detecção de defeitos, e selecionar elementos representativos de cada classe. Exemplos de critérios dessa categoria são *Pairwise Testing* e análise de valor limite.

- **Critério de Cobertura Baseado em Requisitos:** Há casos onde os elementos de um modelo de teste podem ser associados a requisitos do SUT. Por exemplo, identificadores podem ser associados aos requisitos e anotados em transições de um modelo de máquina de estados. A partir disso, casos de teste podem ser avaliados com base na sua capacidade de cobrir essas estruturas anotadas.

- **Especificação de Casos de Teste:** Especificação podem ser definidas em *Ad hoc* para controlar a geração de casos de teste. Além disso, testadores podem usar notações formais para especificar quais casos de teste devem ser gerados. Por exemplo, a execução de um conjunto de caminhos pode ser assegurada por meio de uma especificação que restringa sua cobertura.

- **Critério Estocástico/Aleatório:** Esse tipo de critério é normalmente aplicado na criação de modelos de ambiente e padrões de uso do SUT, tais como a proba-

bilidade de ocorrência de ações ou eventos. Um exemplo de critério deste tipo é o Deterministic State Testing (Avritzer e Larson, 1993).

- **Critério Baseado em Defeitos:** Esse é um tipo de critério normalmente aplicado em modelos de SUT dado o seu objetivo de identificar defeitos no próprio sistema em teste. Teste de mutação é um tipo comum de critério baseado em defeitos.

Tecnologia é a dimensão que diz respeito ao potencial de automação da ferramenta e a forma como os casos de teste são gerados. A partir de um modelo e um critério de seleção, uma abordagem TBM permite gerar casos de teste de modo manual ou automático. Casos de teste podem ser gerados aleatoriamente, usando algorítmos de busca, checagem de modelo (*Model Checking* (Baier e Katoen, 2008)), execução simbólica ou prova de teorema. A seguir, cada uma das tecnologias citadas será brevemente discutida.

- **Geração Manual:** Neste caso, casos de teste são gerados manualmente a partir do modelo de teste do SUT.

- **Geração Aleatória:** Casos de teste podem ser gerados por meio da amostragem aleatória de elementos do espaço de dados de entrada do SUT. Entradas para sistemas reativos podem ser aleatóriamente geradas e aplicadas em um modelo de teste e no próprio SUT para que suas saídas sejam comparadas e validadas.

- **Algorítmos Baseado em Busca:** Algorítmos de busca em grafos e metaheurísticas, tais como algorítmo genético, colônia de formigas e enxame de particulas (Ali et al., 2010), podem ser usados para guiar a geração de testes.

- **Checagem de Modelo:** Checagem de Modelo (do inglês *Model Checking*) é uma tecnologia de geração de testes que visa verificar ou invalidar propriedades de modelos. Usando essa técnica, casos de teste podem ser gerados e usados contra-exemplos a fim de evidenciar propriedades que não estejam sendo satisfeitas. Alcançabilidade é um exemplo de propriedade que diz respeito aos casos onde "eventualmente, um sistema pode alcançar um certo estado ou disparar certas transições".

- **Execução Simbólica:** Essa tecnologia usa modelos de teste executáveis com conjuntos de entradas e restrições para gerar casos de teste ao invés do próprio SUT.

- **Prova de Teorema:** Essa técnica pode ser usada para gerar casos de teste que provem a satisfabilidade de propriedades, tais como restrições definidas em transições de modelos de máquinas de estados.

Assim como ocorre com os critérios de seleção, tecnologias de geração de testes também podem ser combinadas a fim de solucionar problemas mais complexos, tais como a seleção

de dados de teste considerando a existência de transições inalcançáveis (Utting et al., 2012).

Execução de teste é a dimensão que diz respeito ao tempo relativo entre a geração e execução dos casos de teste. Assim como dito na seção anterior, a execução de casos de teste pode ser efetuada basicamente em modo *online* ou *offline*. Há ferramentas que oferecem ambos os modos de execução.

Além dessas dimensões, em um artigo mais recente, Utting et al. (2012) adapta a taxonomia discutida nessa seção considerando uma dimensão denominada *Escopo do Modelo de Especificação*. Essa dimensão delimita os elementos do SUT especificados no modelo de teste que podem incluir somente dados de entrada ou dados de entrada e saída. Modelos de teste especificando somente dados de entrada são geralmente mais fáceis de serem especificados. Modelos contendo entradas e saídas, devido envolverem oráculos de teste, apresentam uma complexidade superior.

3.5 Considerações Finais

Neste capítulo foram introduzidos os conceitos de teste baseado em modelos. As cinco etapas (Modelagem, Geração, Concretização, Execução e Análise) que compõem o processo dessa variante de teste também foram apresentadas e discutidas uma a uma. Uma taxonomia descrevendo os parâmetros que podem ser usados para categorizar abordagens de TBM foi apresentada. Essa taxonomia teve cada uma das suas sete dimensões (Assunto, Independência, Características, Paradigma do Modelo de Teste, Critério de Seleção, Tecnologia de Geração de Testes e Execução de Testes) detalhada. Muito do que foi apresentado aqui será usado no capitulo a seguir que discutirá Teste Baseado em Máquina de Estados Finitos.

Capítulo

4

Teste Baseado em Máquinas de Estados Finitos

4.1 Considerações Iniciais

Segundo Utting et al. (2012) notações baseadas em transição formam uma categoria de modelos que permitem especificar SUTs com base nas transições existentes entre seus diferentes estados. Máquinas de Estados Finitos (MEF) são um tipo de notação baseada em transição permite descrever SUTs com base nos estados do sistema e transições entre eles. Modelos, tais como MEF, têm sido usados desde a década de 80 para projetar e testar sistemas, tais como sistemas embarcados, protocolos de comunicação e circuitos digitais (Broy et al., 2005), e até hoje têm se mostrado ferramentas muito eficientes para a detecção de defeitos.

A fim de fornecer um entendimento sobre Teste Baseado em MEFs, neste capítulo serão discutidos alguns dos principais aspectos teóricos e práticos dessa área. Na Seção 4.2 será exposta a definição formal de MEF, suas características e propriedades e uma visão geral do Teste Baseado em MEFs. Na Seção 4.3 serão discutidos algorítmos de geração de testes para MEFs. Por fim, na Seção 4.4 serão citados exemplos de ferramentas de Teste Baseado em MEFs.

4.2 Máquinas de Estados Finitos

Segundo Simão et al. (2009), uma Máquina de Estados Finitos (MEF) pode ser definida como uma tupla $\mathcal{M} =< S, s_0, I, O, D, \delta, \lambda >$, tal que

- S é um conjunto finito não-vazio de estados;
- $s_0 \in S$ é o estado inicial;
- I é um conjunto finito não-vazio de símbolos de entrada;
- O é um conjunto finito não-vazio de símbolos de saída;
- $D \subseteq S \times I$ é o domínio de especificação;
- $\delta : D \to S$ é a função de transição de estado;
- $\lambda : D \to O$ é a função de saída.

Uma interpretação sugerida por Jonsson (2005) para este conceito de MEF será dada a seguir.

A qualquer momento uma máquina $\mathcal{M}$ encontra-se sempre em um único estado $s \in S$. Esta máquina $\mathcal{M}$ pode receber símbolos de entrada $a \in I$ que levarão $\mathcal{M}$ do estado s para um novo estado $\delta(s, a)$. Além disso, uma saída $\lambda(s, a) = b,\ b \in O$, será gerada pela função de saída de $\mathcal{M}$. O domínio de especificação define os símbolos de entrada válidos para os estados da MEF. Um símbolo de entrada $a \in I$ é dito válido para um estado $s_i \in S$ se (s_i, a).

MEFs podem ser descritas usando tanto grafos direcionados como tabelas de transição de estados (Gill, 1962). Ao usar grafos direcionados para representar MEFs, o conjunto de vértices é formado por todos os estados $s \in S$. Para cada estado $s \in S$ e entrada $a \in I$, podem haver arestas rotuladas com "a/b", tal que $b = \lambda(s, a)$, ligando s ao estado $\delta(s, a)$. Na Figura 4.1 pode ser visto um exemplo de MEF com $I = \{a, b\}$, $O = \{0, 1\}$, $S = \{q0, q1, q2, q3\}$ e $s_0 = q0$ representada na forma de grafo direcionado. Caso uma tabela de transição de estados seja usada para descrever uma MEF, usamos as linhas para descrever os estados e as colunas listando as entradas. Cada célula mostra o resultado das funções de transição de estados e de saída para um estado (linha) e uma entrada (coluna). Na Tabela 4.1 pode ser vista a tabela de transição de estados da MEF da Figura 4.1.

Outra notação usada para descrever MEFs segundo suas funções de transição de estados e funções de saída é "s_i - a/b -> s_j". No exemplo, uma MEF no estado s_i é transferida para s_j e retorna o símbolo de saída b ao receber um símbolo de entrada a, sendo $s_i, s_j \in S, a \in I$ e $b \in O$. Uma sequência finita de símbolos de entrada também pode ser colocada sem a sua respectiva sequência de saída, por exemplo, "s_i - *babaa* -> s_j".

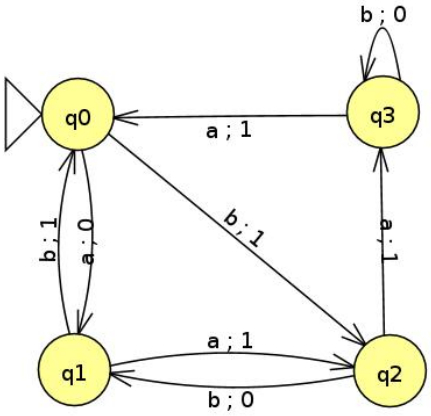


Figura 4.1: MEF Representada como Grafo Direcionado

$s_0 = q0$	λ		δ	
S \ I	a	b	a	b
q0	0	1	q1	q2
q1	1	1	q2	q0
q2	1	0	q3	q1
q3	1	0	q0	q3

Tabela 4.1: MEF Representada como Tabela de Transição de Estados

O conceito de função de transição de estados e função de saída pode ser extendido para o contexto de sequência de símbolos de entrada. Dessa forma, ao aplicar uma sequência de símbolos de entradas $x = a_1 a_2 \dots a_k \in I^*$ em uma MEF $\mathcal{M}$ em $s_1 \in S$ como estado atual dizemos que ela é levada sucessivamente aos estados $s_2, s_3, \dots, s_{k+1}$, onde $s_{i+1} = \delta(s_i, a_i), i = 1, 2 \dots k$, e é gera uma saída $b_1 b_2 \dots b_k \in O^*$, onde $b_i = \lambda(s_i, a_i), i = 1, 2 \dots k$. As função de transição de estados e função de saída também podem ser definidas de modo recursivo, assim na Equação 4.1.

$$\begin{aligned} \delta(s, \epsilon) &= s \\ \lambda(s, \epsilon) &= \epsilon \\ \delta(s, \beta\alpha) &= \delta(\delta(s, \beta), \alpha) \\ \lambda(s, \beta\alpha) &= \lambda(s, \beta)\lambda(\delta(s, \beta), \alpha) \end{aligned} \tag{4.1}$$

No caso da Equação 4.1, a sequência de símbolos $\alpha\beta$ representa a concatenação de duas sequências $\alpha, \beta \in I^*$. A sequência α é dita prefixo de uma sequência ω, denotando $\alpha \leq \omega$, se Os simbolos $\mathcal{M}_S$ e $\mathcal{M}_I$ serão usados, respectivamente, para a especificação de um SUT e para a implementação do SUT ou MEF em teste.$\omega = \alpha\beta$. $\epsilon \in I^*$ representa uma sequência vazia que não altera o estado nem gera saídas ao ser aplicada.

4.2.1 Propriedades e Características

Máquinas de Estados Finitos possuem propriedades e características importantes que no contexto desse trabalho devem ser garantidas (Gargantini, 2005), como:

- **Determinismo:** $\mathcal{M}_S$ e $\mathcal{M}_I$ devem ser *determinísticas*. Uma MEF é dita determinística quando todos os estados tem somente uma única transição para cada entrada. Caso contrário, elas são *não-determinísticas*.

- **Minimalidade:** $\mathcal{M}_S$ deve ser *minimal*, também denominada *reduzida*. Uma máquina *minimal* não apresentam estados equivalentes. Isso garante a existência de ao menos uma sequência capaz de diferenciar cada par de estados por meio da sua saída. Formalmente isso significa $\forall\ s_i, s_j \in S,\ \exists\ \alpha = a_1 a_2 \dots a_k \in I, \lambda(s_i, \alpha) \neq \lambda(s_j, \alpha)$. No caso de uma MEF *não-reduzida*, algorítmos de minimização podem ser aplicados a fim de convertê-las para sua equivalente minimal.

- **Completude:** $\mathcal{M}_S$ deve ser *completamente especificada*, ou *completa*. As funções de saída e transição de estados devem ser totais, isto é, todos os símbolos de entrada de I devem ser válidos em todos os estados de S, ou seja, $D = S \times I$. Caso contrário, a MEF é dita *parcialmente especificada*, ou *parcial*.

- **Conectividade:** $\mathcal{M}_S$ deve ser *fortemente conexa*. Todos os estados devem ser alcançáveis a partir de quaisquer outros estados por meio de uma ou mais transições, ou seja $\forall s_i, s_j \exists \alpha,$ tal que $s_i = \delta(s_j, \alpha)$. Uma máquina é dita *inicialmente conexa* quando possui todos os estados alcançáveis a partir de s_0.

Além destas propriedades, também deve se assumir que para testar $\mathcal{M}_I$ a implementação não muda de comportamento durante o teste e que $\mathcal{M}_I$ possui o mesmo conjunto I e S de $\mathcal{M}_S$. Isso garante que $\mathcal{M}_I$ responda com $b \in O$ a qualquer símbolo de entrada $a \in I$. Sem estas quatro propriedades o teste baseado em MEFs não é possível.

Outras propriedades convenientes de serem garantidas mas não essenciais serão citadas a seguir.

- **Estado Inicial:** $\mathcal{M}_S$ e $\mathcal{M}_I$ estão em um mesmo estado inicial s_0. Entretanto, se $\mathcal{M}_I$ não estiver no mesmo s_0, tipos específicso de sequência permitem conduzir a máquina a um estado específico e podem ser aplicadas a fim de levá-la ao estado s_0.

- **Mesmo Número de Estados:** De acordo com Chow (1978), uma MEF minimal pode apresentar quatro tipos de defeitos: Defeito de Saída, Defeito de Transferência, Defeito de Estado Extra e Defeito de Estado Ausente. Se ambas $\mathcal{M}_I$ e $\mathcal{M}_S$ possuem o mesmo número de estados, somente defeitos de saída e de transferência precisam

ser considerados e defeitos relacionados a adição/remoção de estados podem ser ignorados.

- **Operação *reset*:** $\mathcal{M}_I$ e $\mathcal{M}_S$ possuem uma entrada especial denominada *reset* que permite levar uma MEF ao seu estado inicial após a aplicação qualquer sequência. Formalmente, uma operação *reset* é definida como $\forall\ s \in S, \delta(s, reset) = s_0$. Em MEFs *inicialmente conexas* a operação de *reset* é útil pois permite restaurar o seu estado mesmo quando estados de *deadlock* são alcançados. Alguns algorítmos de teste dependem de operações de *reset*.

- **Operação *status*:** $\mathcal{M}_I$ e $\mathcal{M}_S$ possuem uma entrada especial denominada *status* que permite consultar seu estado atual. Formalmente, uma operação *status* pode ser definida como $\forall s_i \in S,\ \lambda(s_i, status) = s_i$.

- **Operação *set*:** $\mathcal{M}_I$ e $\mathcal{M}_S$ possuem uma entrada especial denominada *set* que permite transportar uma MEF para um estado $s_i \in S$. Formalmente, uma operação *set* pode ser definida como $\forall s_i, s_j \in S,\ \delta(s_i, set(s_j)) = s_j$. A operação *set* não gera saída.

4.2.2 Teste Baseado em Máquinas de Estados Finitos

Segundo Lee e Yannakakis (1996), o objetivo do Teste Baseado em Máquinas de Estados Finitos (TBMEF) pode ser definido da seguinte forma:

> Dada uma máquina de estados finitos $\mathcal{M}_S$ usada como especificação no qual temos conhecimento do seu diagrama de transição, e outra máquina $\mathcal{M}_I$ no qual apenas podemos observar o seu comportamento, deseja-se testar se $\mathcal{M}_I$ implementa corretamente ou está em conformidade com $\mathcal{M}_S$.

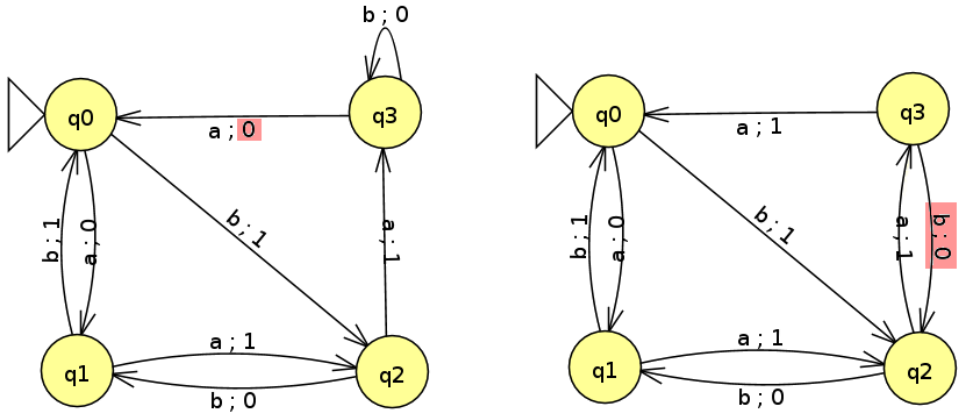
Considerando uma máquina $\mathcal{M}_S$ minimal com n estados, dizemos que $\Im(\mathcal{M}_S)$ denota o conjunto de todas as MEFs determinísticas, completas, minimais, com o mesmo conjunto de símbolos de entrada e saída. Dessa forma, segundo (Chow, 1978), podemos dizer que o TBMEF visa encontrar um conjunto de sequências de entrada que permita diferenciar $\mathcal{M}_S$ de todas as demais MEFs $\mathcal{M} \in \Im(\mathcal{M}_S)$ que apresentem algum dos seguintes defeitos:

- **Defeito de Operação:** $\mathcal{M}_\mathcal{I}$ tem um defeito de operação se ela puder se tornar equivalente a $\mathcal{M}_S$ a partir da modificação de uma saída em uma transição, sem adicionar ou remover estados.

- **Defeito de Transferência:** $\mathcal{M}_\mathcal{I}$ tem um defeito de transferência se ela puder se tornar equivalente a $\mathcal{M}_S$ a partir da modificação da função de transição de estados, sem adicionar ou remover estados.

- **Estado Extra:** $\mathcal{M}_{\mathcal{I}}$ tem um defeito de estado extra se ela puder se tornar equivalente a $\mathcal{M}_S$ a partir da remoção de um estado.
- **Estado Ausente:** $\mathcal{M}_{\mathcal{I}}$ tem um defeito de estado extra se ela puder se tornar equivalente a $\mathcal{M}_S$ a partir da adição de um estado.

Na Figura 4.2, são exemplificados defeitos de operação e de transferêcia que podem afetar uma MEF como a da Figura 4.1.



(a) Defeito de Operação **(b)** Defeito de Transferência

Figura 4.2: Exemplos de Defeitos em MEFs

Entretanto, como o conjunto $\Im(\mathcal{M}_S)$ e I^* é muito grande, abordagens exaustivas de teste se tornam inviáveis. Em consequência disso, medidas alternativas devem ser usadas para sistematicamente gerar testes que diferenciem com um certo grau de certeza $\mathcal{M}_S$ de todas as demais MEFs não-equivalentes.

Duas MEFs $\mathcal{M}_S =< S^S, s_0^S, I, O, D^S, \delta^S, \lambda^S >$ e $\mathcal{M}_I =< S^I, s_0^I, I, O^I, D^I, \delta^I, \lambda^I >$ são ditas equivalentes se, para cada estado de $\mathcal{M}_S$, existe um estado equivalente em $\mathcal{M}_I$, ou seja, $\forall\ s^S \in S^S,\ \exists\ s^I \in S^I\ \mid s^S \equiv s^I$. Dois estados s^S e s^I são equivalentes se gerarem as mesmas saídas para todas as entradas, ou seja, $s^S \equiv s^I \iff \forall\ \alpha \in I,\ \lambda(s^S, \alpha) = \lambda(s^I, \alpha)$.

Para executar cada sequência de um conjunto de testes T a máquina $\mathcal{M}$ precisa ter seu estado inicial restaurado. Para isso, uma operação de *reset* precisa ser executada antes de qualquer sequência ser aplicada, representada normalmente por meio do símbolo r. Ao ser executada em uma MEF, uma sequência de testes apresenta um *custo de execução* que pode ser medido pelo tamanho da sequência somado a operação de *reset*. Por exemplo, se considerarmos uma sequência $\alpha = babaa$, o custo de execução de α será $\mid \alpha \mid +1 = 6$.

4.3 Metodos de Geração de Casos de Teste

Geração de Casos de Teste consiste na etapa do TBM onde são selecionados os critérios de seleção e tecnologias de geração de testes a serem usados para obter casos de teste a partir do modelo de especificação do SUT (Utting et al., 2012). Em TBMEF, há uma série de métodos de geração que apresentam diferentes características e restrições de uso. Estes métodos usam sequências especiais, denominadas Sequências Básicas, para obter resultados parciais que auxiliam no processo de geração de casos de teste. A seguir, na Seção 4.3.1, serão apresentadas algumas das principais sequências básicas usadas em TBMEF. Essas sequências dão apoio para muitos dos métodos de geração de sequências mais clássicos e mais modernos que serão apresentados na Seção 4.3.2.

4.3.1 Sequências Básicas

Métodos de geração de teste para TBMEF são fortemente baseados em conjuntos de sequências denominadas sequências básicas. Estas sequências são importantes pois fornecem resultados parciais que auxiliam os métodos de geração de sequências a atingirem seus objetivos. A seguir serão apresentadas algumas das principais sequências básicas que dão apoio a muitos algorítmos de geração de sequências de teste.

4.3.1.1 State Cover (Q)

O conjunto *State Cover* é um conjunto Q de n sequências de entrada que permite levar uma máquina $\mathcal{M}$ a todos os seus n estados. Formalmente, ele pode ser definido como $\forall\ s \in S,\ \exists x \in Q \mid \delta(s_0, x) = s$. Considerando a MEF da Figura 4.1, temos o seguinte conjunto *State Cover*: $Q = \{\epsilon, a, b, ba\}$.

4.3.1.2 Transition Cover (P)

O conjunto *Transition Cover* é um conjunto P de sequências que permite atravessar todas as transições de uma MEF. Ele pode ser gerado a partir do conjunto Q por meio da concatenação das sequências $x \in Q$ com as entradas válidas para cada estado alcançado usando x. Considerando a MEF da Figura 4.1, temos o seguinte conjunto *Transition Cover*: $P = \{a, b, aa, ab, ba, bb, baa, bab\}$.

4.3.1.3 Sequência de Sincronização (SS)

Antes da aplicação de uma sequência de testes é importante que o estado de uma MEF seja restaurado para s_0. Neste caso, operações de *reset* confiáveis podem ser de grande utilidade porém nem todos os sistemas podem dispor de tal operação. Para contornar isso,

sequências que permitam restaurar uma MEF para um dado estado podem ser usadas, tal como a Sequências de Sincronização.

Uma Sequências de Sincronização permite que uma MEF seja encaminhada para um único estado $s_j \in S$ independentemente do seu estado atual. Vale ressaltar que uma SS não depende das saídas de uma MEF e que há casos onde ela pode não existir para uma dada MEF. Considerando a máquina de estados da Figura 4.3, temos as seguintes sequências de sincronização: $SS_{q_0} = abbaaa$, $SS_{q_1} = abb$, $SS_{q_2} = abba$ e $SS_{q_3} = abbaa$.

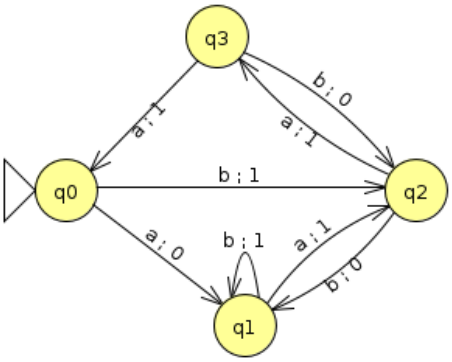


Figura 4.3: Máquina de Estados com Sequência de Sincronização

4.3.1.4 Sequência de Separação

Uma Sequência de Separação consiste de uma sequência $x \in I$ que permite diferenciar um par de estados s_i e s_j por meio das suas saídas, ou seja, $\lambda(s_i, x) \neq \lambda(s_j, x)$. Considerando a MEF da Figura 4.1, temos $x = a$ como Sequência de Separação para o par de estados q_0 e q_1 pois $\lambda(q_0, a) = 0$ e $\lambda(q_1, a) = 1$.

4.3.1.5 Sequência de Distinção (DS)

Uma Sequência de Distinção (do inglês, *Distinguishing Sequence* - DS) é uma sequência de simbolos de entrada que produz saídas distintas para cada um dos estados de uma MEF permitindo a identificação do seu estado inicial, ou seja, uma sequência $x \in I$ é dita uma DS se $\forall\ s_i, s_j \in S, \lambda(s_i, x) \neq \lambda(s_j, x)$.

Sequências de Distinção podem ser adaptativas (*Adaptive Distinguising Sequence* - ADS), se símbolos são definidos com base nas saídas anteriores, ou pré-definida (*Preset Distinguising Sequence* - PDS), se definidas previamente. Sequências adaptativas são geralmente apresentadas na forma de árvores de decisão (Broy et al., 2005).

Considerando a MEF da Figura 4.1, temos "*bb*" como uma PDS pois, $\lambda(q_0, bb) = 10$, $\lambda(q_1, bb) = 11$, $\lambda(q_2, bb) = 01$ e $\lambda(q_3, bb) = 00$; e na Figura 4.4 uma ADS.

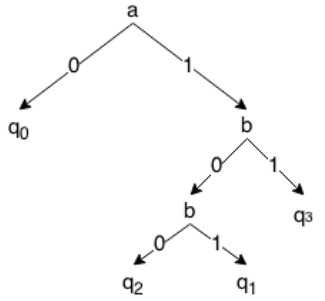


Figura 4.4: Exemplo de Sequência de Distinção Adaptativa

4.3.1.6 Sequência UIO

Uma Sequência Única de Entrada-Saída (do inglês, *Unique Input-Output Sequence* - UIO) é uma sequência capaz de diferenciar um dado estado $s_i \in S$ de todos os demais estados $s_j \in S \mid s_i \neq s_j$. Considerando a MEF da Figura 4.1, as seguintes sequências são UIO: $UIO_{q_0} = \{a/0\}$, $UIO_{q_1} = \{b/1, b/1\}$, $UIO_{q_2} = \{b/0, b/1\}$ e $UIO_{q_3} = \{b/0, b/0\}$.

4.3.1.7 Conjunto de Caracterização (W)

Um conjunto de caracterização, também denominado conjunto W, é composto por sequências de entrada capazes de identificar em conjunto o estado original de uma MEF. Considerando a MEF da Figura 4.1 e a Tabela 4.2, podemos ver que $W = \{bb\}$ consiste de um conjunto de caracterização capaz de diferenciar todos os estados.

S \ I	bb
q0	10
q1	11
q2	01
q3	00

Tabela 4.2: Exemplo de Conjunto W

4.3.1.8 Conjunto de Identificadores Harmonizados (F)

O Conjunto de Identificadores Harmonizados, também denominado Família de Separação, é um conjunto de identificadores H_i para cada estado $s_i \in S$ de uma MEF. O conjunto H_i é composto por sequências de entrada que satisfazem as seguintes condições: $\forall s_i, s_j \in S$, $\exists \beta \in H_i$ e $\gamma \in H_j$, tal que existe um prefixo α comum a β e γ e $\lambda(s_i, \alpha) \neq \lambda(s_j, \alpha)$. Um Conjunto de Identificadores Harmonizados sempre existe para MEFs minimais. Considerando a MEF da Figura 4.1 temos o seguinte Conjunto de Identificadores $F = \{H_1, H_2, H_3, H_4\}$ com $H_1 = \{bb\}$, $H_2 = \{bb\}$, $H_3 = \{bb\}$ e $H_4 = \{bb\}$.

4.3.2 Métodos de Geração de Sequências

Nesta seção serão apresentados os métodos TT, DS, UIO, W, Wp, HSI, H, P e SPY de geração de sequências para Teste de MEFs. Esses métodos vem sendo estudados na teoria e na prática por diversos autores, tais como Dorofeeva et al. (2010, 2005b); Endo e Simao (2013); Simão et al. (2009), interessados em obter experimentalmente dados que evidenciem informações como característica dos seus conjuntos de teste, tais como total e comprimento de casos de teste; o seu custo, considerando comprimento de todas as sequências de um conjunto de teste; e efetividade, no que tange sua taxa de detecção de defeitos. Estudos experimentais permitem obter dados empíricos que possibilitem comparar vários métodos em diferentes circunstâncias de modo que a seleção de um método seja feita mais conscientemente.

4.3.2.1 Método TT

O método *Transition Tour* (TT) é relativamente simples, quando comparado com os demais métodos de geração. O conjunto de testes gerado pelo método TT permite atravessar todas as transições de uma MEF e, em consequência disso, detectar defeitos de operação porém não garante a cobertura de defeitos de transferência. Um conjunto TT é formado por sequências que saem do estado s_0 e percorrem todas as transições. O tamanho do conjunto de testes depende da conectividade da MEF testada. No caso de uma MEF fortemente conexa, o conjunto será composto por apenas uma sequência, caso contrário operações de *reset* tornam-se necessárias para restaurar o estado da MEF para que novas sequências sejam aplicadas.

4.3.2.2 Método DS

O método DS usa sequências de distinção e sequências de sincronização para gerar testes, em consequência disso, sua aplicação fica dependente da existência dessas sequências. Para aplicar o Método DS, inicialmente deve ser selecionada uma sequência DS que deve ser preferencialmente a mais curta, pois ela será usada várias vezes. A sequência DS é usada para gerar um grafo denominado X_d com um nó para cada estado da MEF e com transições apontando para estado final após a aplicação da sequência DS e com suas respectivas saídas. O grafo X_d da MEF da Figura 4.3 pode ser visto na Figura 4.5.

Após a geração do grafo X_d, devem ser geradas duas subsequências denominadas sequência-α e sequência-β. Estas sequências permitem verificar, respectivamente, todos os estados da MEF e todas as transições.

A sequência-α é gerada percorrendo o grafo X_d sem repetir arestas. Um estado inicial (origem) sem arestas de chegada deve ser primeiramente selecionado e, ao aplicar a sequên-

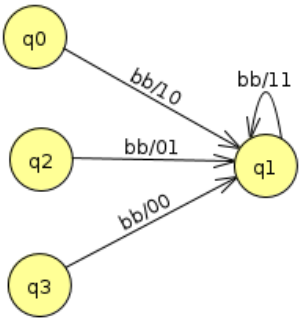


Figura 4.5: Exemplo de Grafo X_d

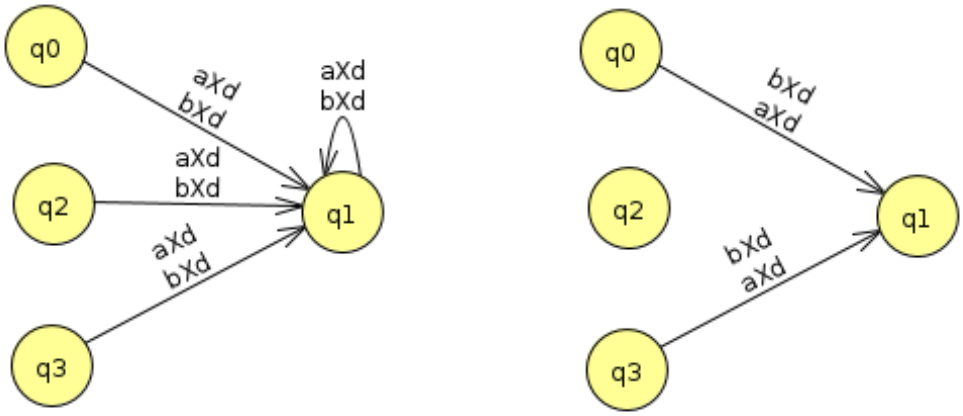
cia DS, os estados percorridos devem ser marcados como "reconhecidos" conforme forem sendo atingidos. Quando um estado reconhecido anteriormente é alcançado, a sequência DS deve ser aplicada novamente a fim de garantir que o ultimo estado foi de fato alcançado. Após isso, deve ser aplicada uma sequência que leve do ultimo estado alcançado até um novo estado ainda não reconhecido. Com isso, a sequência-α obtida do grafo X_d da Figura 4.5 é: $< b, b, b, b, a, b, b, a, a, b, b >$.

A sequência-β permite verificar as transições de uma MEF. Para isso, deve ser gerado um diagrama denominado Diagrama-β (Figura 4.6(a)) que tem como nós os estados e arestas a concatenação dos símbolos de saída válidos para um nó de origem com a sequência DS escolhida, $x_i X_d \mid x_i \in I$. A geração de caminhos que atravessem esse diagrama permitirá a cobertura de todas as transições da sua respectiva MEF. Entretanto, o Diagrama-β pode ser minimizado, se considerarmos que a última transição da aplicação das sequências DS verifica o seu estado de destino. Logo, as transições referentes ao ultimo símbolo da sequência DS dos estados "reconhecidos" podem ser eliminadas. O Diagrama-β Minimizado do Diagrama-β em questão pode ser visto na Figura 4.6(b). Apartir deste Diagrama-β Minimizado e do método descrito anteriormente podemos gerar a seguinte sequência-β: $< a, X_d, a, a, b, X_d, a, a, a, a, X_d, a, a, a, b, X_d >$.

4.3.2.3 Método UIO

O método UIO usa o conjunto *State Cover* e de sequências UIO para testar máquinas de estado. Para cada sequência $Q(s_i)$ do conjunto *State Cover* que leva a MEF ao estado $s_i \in S$, sao concatenados um símbolo $x \in I \mid \delta(s_i, x) = s_j$. Após isso, cada sequência é concatenada com sua respectiva UIO_{s_j} que verifica o estado s_j alcançado. Dessa forma, podemos definir as sequências do Método UIO como: $\forall s_i \in S, x \in I \rightarrow Q(s_i) \cdot x \cdot UIO_{s_j}$.

Considerando a MEF da Figura 4.1, temos para estado e entrada da MEF o seguinte conjunto de sequências gerado pelo método UIO: $UIO(q_0, a) = abb$, $UIO(q_0, b) =$



(a) Diagrama-β **(b)** Diagrama-β minimizado

Figura 4.6: Diagramas β usados pelo Método DS

bbb, $UIO(q_1, a) = aabb$, $UIO(q_1, b) = abbb$, $UIO(q_2, a) = babb$, $UIO(q_2, b) = bbbb$, $UIO(q_3, a) = baabb$ e $UIO(q_3, b) = babbb$.

4.3.2.4 Método W

O Método W, como popularmente ficou conhecido o *Automata Theoretic Method*, é um dos métodos mais clássicos de geração de sequências de teste (Chow, 1978). O Método W se baseia no uso dos conjuntos *Transition Cover* P, que permite percorer as transições da MEF em teste, e no Conjunto de Caracterização W, para identificar os estados alcançados. Essencialmente o método W consiste da execução dos seguintes passos:

1. **Geração do Conjunto P:** O conjunto P é gerado por meio de uma estrutura denominada árvore de teste. Uma árvore de teste consiste de uma árvore onde seus nós são nomeados com os estados da MEF e suas arestas com as entradas válidas do seu respectivo estado. Caminhos em uma árvore de teste permitem atravessar sem repetição as transições de uma MEF.

2. **Geração do Conjunto W:** Um conjunto de caracterização é gerado a fim de permitir a identificação de cada um dos estados de uma MEF. O uso do conjunto W permite a identificação de defeitos de transferência.

3. **Estimativa de Estados Extras:** Essa é uma etapa decisiva para o sucesso do Método W pois ela é responsável por balancear o esforço realizado para identificar defeitos de estados extras. Um valor m é estimado para definir o total de estados que uma $\mathcal{M}_I$ possui, a contar os estados extras, considerando n como o total de estados

de uma especificação $\mathcal{M}_S$. Esse valor é usado para gerar sequências de tamanho limitado a $m - n$ que combinem os símbolos de $x \in I$. Formalmente definimos esse conjunto como $\cup_{i=0}^{m-n}(I^i)$.

4. **Geração do Conjunto W':** O conjunto W' é formado pela concatenação do Conjunto *Transition Cover* P, conjunto das sequências de tamanho limitado a $m - n$ e o Conjunto de Caracterização W, ou seja, $W' = P \cdot \cup_{i=0}^{m-n}(I^i) \cdot W$.

Vale ressaltar que se considerarmos m=n, ou seja, um $\mathcal{M}_I$ sem defeito de estados extras, teremos $W' = W$. Considerando a MEF da Figura 4.1, temos a árvore de testes ilustrada na Figura 4.7 com o dado conjunto $W' = W$ tal que $m = n$: $W' = \{aabb, abbb, baabb, babbb, bbbb\}$.

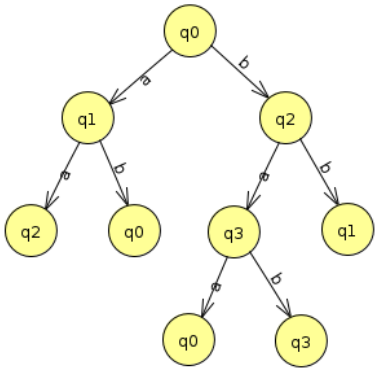


Figura 4.7: Exemplo de Árvore de Teste

4.3.2.5 Método Wp

O Método Wp (*partial W*) é uma melhoria do Método W que busca gerar sequências de teste menores, com a mesma garantia de cobertura de erros do seu antecessor. O termo "*partial*" se deve ao uso de um subconjunto de W para identificar os estados de uma MEF, ao invés do seu uso integral. O Método Wp é composto de duas fases:

1. **Verificação de Estados:** Um conjunto denominado C_1 é gerado por meio da concatenação dos Conjuntos *State Cover* e de Caracterização, ou seja, $C_1 = Q \cdot W$.

2. **Verificação de Transições Não-Cobertas:** Com a aplicação de C_1, uma parcela das transições já será coberta, o que torna desnecessário o uso de todo o conjunto *Transition Cover*. A partir disso, um conjunto $R = P - Q$, composto pelas transições de P não cobertas por Q, pode ser usado para verificar as transições de uma MEF sem afetar a capacidade de detecção de defeitos. Os elementos do conjunto R são concatenados com o conjunto de caracterização W considerando seus respectivos

W_i, o que é denominado $C_2 = R \otimes W$. Em W_i, i consiste do índice do estado que uma MEF atinge ao ser aplicada uma dada sequência de R.

Considerando a MEF da Figura 4.1, teremos os seguintes conjuntos ao aplicar o método Wp: $C_1 \cup C_2 = \{aabb, abbb, baabb, babbb, bbbb\}$. Vale ressaltar que o conjunto apresentado foi reduzido através da remoção de sequências identificadas como prefíxo de outras sequências. Essa remoção pode ser feita pois a aplicação de uma sequência também garante a verificação dos seus prefíxos.

4.3.2.6 Método HSI

O método HSI (do inglês, *Harmonized State Identification*), proposto por Luo et al. (1995), é um algorítmo de geração de sequências de teste aplicável tanto em MEFs completas como parciais. Esse método usa o conjunto de identificadores harmonizados para distinguir os estados de uma MEF e pode ser descrito em duas fases:

1. **Verificação de Estados:** Nesta etapa, o conjunto *State Cover* Q é concatenado com o conjunto de identificadores harmonizados.

2. **Verificação de Transições:** Nesta etapa, as transições não abrangidas durante a primeira etapa são verificadas através da concatencação do conjunto *Transition Cover* com seus respectivos identificadores harmonizados.

Considerando a MEF da Figura 4.1 e o processo citado acima, teremos o seguinte conjunto ao aplicar o método HSI, com prefíxos já removidos: $\{aabb, abbb, baabb, babbb, bbbb\}$.

4.3.2.7 Método H

O método H, assim como o HSI, também usa o conjunto de identificadores harmonizados para distinguir os estados de uma MEF (Dorofeeva et al., 2005a). Entretanto, diferentemente do HSI que realiza previamente a seleção dos identificadores, o método H seleciona os identificadores dinâmicamente, ou *on-the-fly* (Dorofeeva et al., 2005a). Estudos experimentais, tais como Dorofeeva et al. (2005a); Endo e Simao (2013), mostram que o método H permite a obtenção de sequências de menor tamanho (cerca de 66%) em comparação ao HSI.

4.3.2.8 Método P

O método P, proposto por Simão e Petrenko (2009a), é um método que pode ser usado tanto para gerar conjuntos de teste como incrementar conjuntos pré-existentes. Da mesma forma como o método H, o método P também seleciona sequências *on-the-fly*. Basicamente, o método P é executado em duas etapas:

1. **Distinção de Estados:** Um conjunto capaz de distinguir os estados da MEF é gerado.

2. **Condições de Sufuciência:** Transições não cobertas são verificadas através da condição de suficiência dos casos de teste. Condições de suficiência fornecem um meio de verificar se um dado conjunto de testes é m-completo, ou seja, se ele é capaz de detectar todos os defeitos de um domínio (Endo e Simao, 2013).

4.3.2.9 Método SPY

O método SPY (Simão et al., 2012) é um algorítmo de geração de sequências que utiliza uma estratégia que permite reduzir o tamanho de conjuntos de teste. É um algorítmo que gera sequências de teste *on-the-fly* a fim de evitar a inclusão de ramos no conjunto de testes.

4.4 Ferramentas de Teste Baseado em Modelos

Automação é uma caracteristica inerente do Teste Baseado em Modelos. Neste contexto, o uso de ferramentas que auxiliem a geração e execução de testes favorecem a adoção de técnicas mais avançadas de teste, tal como o Teste Baseado em Máquinas de Estados.

Shafique e Labiche (2013) apresentam uma revisão sistemática onde uma série de ferramentas de teste baseado em estados são apresentadas e caracterizadas segundo diferentes parâmetros, tais como critérios de seleção disponibilizados, geração de oráculos de teste, código executável e apoio a atividades relacionadas ao teste. Nesse trabalho, um total de 12 ferramentas é apresentado, sendo 3 classificadas como código aberto e baseadas em máquinas de estados. Essas três ferramentas serão discutidas a seguir.

GraphWalker[1] é uma ferramenta código aberto desenvolvida em Java que disponibiliza diferentes algorítmos de geração de teste que podem ser aplicados a modelos de máquinas de estados especificados usando em uma linguagem denominada GraphML [2]. A ferramenta não dispõe de uma interface gráfica, entretanto o software de modelagem yEd[3] pode ser usado para auxiliar no processo de criação dos modelos.

NModel[4] é um framework de código aberto desenvolvido em C# que permite gerar casos de teste a partir de classes especificadas em C#. O framework NModel inclui uma biblioteca para especificar modelos de programas usando C#, uma ferramenta de análise e visualização denominada mpv (*Model Program Viewer*), uma ferramenta de geração de

[1] `http://graphwalker.org/`
[2] `http://graphml.graphdrawing.org/`
[3] `http://www.yworks.com/en/products/yfiles/yed/`
[4] `http://nmodel.codeplex.com/`

testes denominada otg (*Offline TestGenerator*) e uma ferramenta de execução de testes *online* denominada ct (*Conformance Tester*). O NModel também podem ser usado no desenvolvimento de ferramentas de teste customizadas.

PyModel[5] é um framework Python para teste baseado em modelos de máquinas de estados. PyModel pode ser usado para testar protocolos de comunicação, *sockets*, sistemas embarcados, aplicações web e com paralelismo. Ele pode gerar testes *offline*, como testes unitários ou *online*, o que é interessante no caso do teste de sistemas com comportamento não-determinístico. PyModel é composto por 3 programas: PyModel analyzer (pma), que permite gerar MEFs e analisar propriedades por meio da análise dos modelos; PyModel graphics (pmg), que permite gerar arquivos no formato *.dot* permitindo a visualização das MEFs; e PyModel tester (pmt) mostra informações sobre a execução de testes (*trace*), gera e executa testes em modo *offline* e *online*.

Um ponto ressaltado por Shafique e Labiche (2013) é que uma parte significativa das ferramentas identificadas apenas fornece critérios de seleção de testes mais simples, tais como cobertura de estados e transições, e que poucos deles disponibilizam critérios mais complexos, tal como pares de transição.

Além dessas ferramentas, Proteum/FSM, Plavis/FSM e JPlavis/FSM são outros exemplos de ferramentas para Teste Bbaseado em MEFs.

Proteum/FSM (Fabbri et al., 1999) é uma ferramenta de teste e validação de máquinas de estados finitos usando teste de mutação. A ferramenta dispõe de uma interface gráfica que permite explorar e aprender conceitos de teste de mutação de uma maneira fácil e controlada e de uma interface de linha de comando para usuários mais avançados que preferirem utilizar *scripts* para automatizar tarefas de teste.

Plavis/FSM (do inglês, PLAtform for Software Validation Integration on Space Systems) é uma ferramenta de teste que tem o objetivo de disponibilizar mecanismos de geração automática de casos de teste a partir de especificações em máquinas de estados finitos (Simão et al., 2005). Este projeto foi realizado em parceria com a Universidade de São Paulo (USP), Universidade Federal de São Carlos (UFSCar), Instituto Nacional de Pesquisas Espaciais (INPE), Universidade Estadual de Ponta Grossa (UEPG), Universidade Estadual de Campinas (UNICAMP) e Centro Universitário Eurípides de Marília (Univem). Ela foi desenvolvida em plataforma web que integra a ferramenta Proteum/FSM em um ambiente onde diversos métodos tais como Método W, Switch-Cover e UIO podem ser aplicados a fim de gerar casos de teste.

JPlavis/FSM (Pinheiro, 2012) é um sistema de geração de testes baseado em MEFs desenvolvido em Java para desktop, adaptação da sua antecessora Plavis/FSM, que visa maior portabilidade da ferramenta e facilidade de instalação. Diversas melhorias foram

[5] `http://staff.washington.edu/jon/pymodel/www/`

implementadas, tais como uma nova interface gráfica para construção de MEFs assim como uma série de outros métodos mais modernos de geração de testes.

4.5 Considerações Finais

Neste capítulo foram introduzidos conceitos e definições que fundamentam o Teste Baseado em Máquinas de Estados Finitos. A definição de máquina de estados finitos e as suas propriedades e características básicas foram apresentadas. Algumas das principais sequências básicas e métodos de geração de sequências de teste para máquinas de estados finitos também foram apresentos junto de alguns exemplos. Os principais conceitos discutidos foram aplicados em exemplos de máquinas de estados visando mostrar os resultados que podem ser obtidos através do teste de MEFs. Por fim, uma lista de ferramentas de código aberto e desenvolvidas no contexto de projetos acadêmicos foi apresentada visando fornecer um conjunto de softwares que possam ser usados em etudos experimentais, tal como o proposto neste trabalho de mestrado.

Capítulo

5

Controle de Acesso

5.1 Considerações Iniciais

Ataques cibernéticos têm se tornado potencialmente mais desastrosos, uma vez que a depedência em tecnologia da informação da sociedade atual aumenta. Junto disso, a área de segurança computacional tem recebido maior atenção da comunidade de engenharia de software e de organizações de pequeno, médio e grande porte.

De acordo com o Guttman e Roback (1995), segurança computacional consiste na proteção concedida a um sistema de informação a fim de preservar os requisitos de Confidencialidade, Integridade e Disponibilidade, também denominados Tríade CIA (do inglês, ***C****onfidentiality*, ***I****ntegrity* e ***A****vailability*), dos seus recursos de hardware, software, firmware, informação, dados e telecomunicação.

Confidencialidade diz respeito à garantia de que informações sensíveis sejam acessadas somente por pessoal autorizado. Integridade refere-se à garantia de que informações não sejam modificadas de modo despercebido por pessoal não autorizado. Disponibilidade envolve a tarefa de assegurar que um sistema forneça seus serviços aos usuários em tempo integral. Para atender a estes requisitos, diversos mecanismos de segurança podem ser implementados como estratégia de defesa contra essas ameaças (Stallings e Brown, 2007).

Controle de acesso lógico, ou simplesmente Controle de Acesso (Nakamura e de Geus, 2007), é um dos meios mais comuns de se garantir a segurança de dados confidenciais armazenados em discos ou servidores, transferidos por redes de computadores ou que

sejam acessados e modificados por usuários, também denominados respectivamente *data at-rest*, *data in-motion* e *data in-use* (Shabtai et al., 2012). Para que isso seja garantido, um sistema deve contar com mecanismos que controlem todo o acesso a ele e aos seus recursos, habilitando-o somente ao pessoal autorizado. Estes mecanismos implementam o controle imposto pelos modelos e políticas de segurança estabelecidas para o sistema.

Nesse capítulo serão discutidos conceitos gerais sobre controle de acesso, mecanismos de controle de acesso, modelos de controle de acesso e alguns exemplos de software para controle de acesso.

5.2 Terminologia da Segurança da Informação

Para descrever o contexto da segurança da informação em uma organização, será usada a terminologia proposta por Stallings e Brown (2007). De acordo com ela, uma organização conta basicamente com três atores físicos (organização, atacantes e ativos organizacionais) e quatro atores lógicos (mecanismos de segurança, vulnerabilidades, ameaças e riscos de segurança). Na Figura 5.1, adaptada de (Shabtai et al., 2012), os termos descrevendo os atores físicos e lógicos podem ser visualizados, respectivamente em caixas cinzas e brancas, junto das relações entre eles.

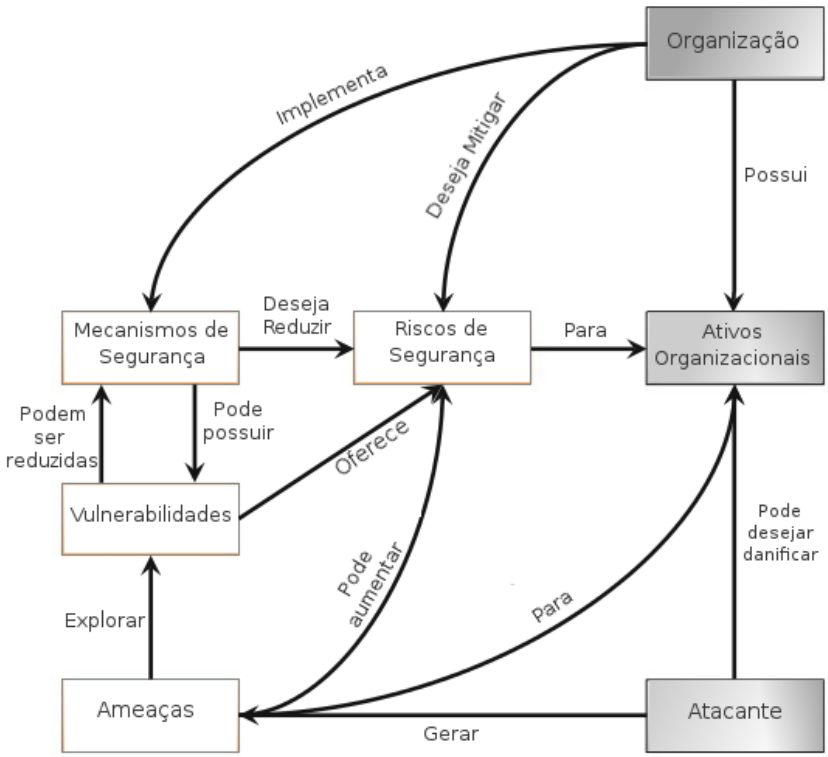


Figura 5.1: Terminologia da Segurança da Informação - Atores e Relações

Uma organização possui inúmeros ativos organizacionais com alto valor e que, em consequência disso, exijem um maior controle durante o seu uso, armazenamento e transferência. Para isso, mecanismos de segurança podem ser implementados a fim de mitigar, detectar e reduzir riscos e vulnerabilidades (falhas de sistema) que possam comprometer a segurança destes ativos. Estas ameaças podem se originar em usuários mal intencionados, também denominados atacantes, que desejem danificar sistemas explorando falhas existentes neles ou até mesmo nos próprios mecanismos de segurança (Shabtai et al., 2012).

Vulnerabilidades que possam ameaçar a segurança de sistemas, entretanto, podem também ser tratadas ainda durante o processo de desenvolvimento por meio de contramedidas. Abordagens, como codificação segura, ofuscação de código e métodos formais estão entre algumas das principais técnicas. Em especial, métodos formais destacam-se neste contexto por permitirem uma verificação eficiente e precisa de modo que seja garantido sistemática e formalmente a inexistência de defeitos em componentes de software. Apesar dos métodos formais em geral serem falhos no que tange sua usabilidade, elas permitem explorar projetos de software exaustivamente e muitas vezes identificar vulnerabilidades críticas (Jang-Jaccard e Nepal, 2014), o que é mandatório em sistemas de missão crítica. No Capítulo 6 abordagens de Teste de Controle de Acesso Baseado em Modelos serão discutido em detalhes.

5.3 Conceitos Básicos de Controle de Acesso

Ferramentas de controle de acesso podem ser divididas em controle de acesso lógico e controle de acesso físico. Ferramentas de controle de acesso lógico são usadas para credenciamento, validação, autorização e atribuição de responsabilidades em uma infraestrutura e nos sistemas que a compõem. Elas podem implementar medidas de segurança para sistemas operacionais, sistemas de administração de infraestrutura, bancos de dados, processos entre outros tipos de aplicações e ativos organizacionais. Ferramentas de controle de acesso físico, por sua vez, constituem artifícios mecânicos que intervêem no acesso físico a um local usando, por exemplo, chaves. Vale ressaltar que nem sempre a diferença entre ferramentas de controle de acesso físico e lógicos é clara (Collins, 2013). Ferramentas de controle de acesso físico, como fechaduras eletrônicas, são controladas por softwares que lêem *chips* de cartões e liberam o acesso somente aqueles indivíduos apropriadamente autorizados. Desse modo, ferramentas de controle de acesso físico também podem ser considerados ferramentas de controle de acesso lógico.

Uma vantagem que ferramentas de controle de acesso lógicos apresentam, quando comparadas com as ferramentas de controle de acesso físicos é a facilidade de instantâneamente

revogar ou alterar permissões atribuidas a usuários. Por exemplo, um empregado pode ter seu cartão de identificação desabilitado automaticamente ao ser demitido ou ter novas permissões habilitadas ao ser promovido para um novo cargo. Ferramentas de controle de acesso têm sido usadas em diversos contextos, como hospitais, *data centers*, organizações governamentais, empresas privadas e departamentos de polícia.

O controle de acesso pode utilizar diferentes fatores de identificação. Fatores baseados na propriedade consistem de elementos tangíveis que um usuário pode possui em mãos, como cartões de identidade, *tokens* ou telefones. Fatores baseados no conhecimento utilizam-se de partes de informação no qual o usuário a ser identificado tem ciência, como uma senha, código PIN ou resposta para uma pergunta de segurança. Fatores baseados em características físicas fundamentam-se em traços físicos de propriedade de um indivíduo, como impressão digital, padrões na retina, na voz, assinatura ou outros identificadores biométricos (Collins, 2013).

5.4 Mecanismos de Controle de Acesso

De acordo com Jaeger (2011), um mecanismo de controle de acesso deve funcionar como um monitor de referência. Além disso, deve atender a um conjunto de requisitos para que possa ser considerado de fato um mecanismo de validação de referência. Um monitor de referência deve ser implementado de modo que ele seja pequeno, à prova de falsificação, não-burlável e confinado em um núcleo de segurança. Desta forma, garante-se que modificações em um mecanismo de controle de acesso não sejam facilmente implementadas, passem despercebidas e torna-o suscetível a métodos rigorosos de verificação formal (Samarati e Vimercati, 2001).

Samarati e Vimercati (2001) sugerem que o processo de desenvolvimento de mecanismos de controle de acesso seja implementado usando uma abordagem multi-fase baseada nos seguintes conceitos:

- **Política de Segurança:** Define as regras em alto nível que definirão como o controle de acesso deverá ser regulamentado. Estas regras também costumam ser chamadas de políticas. São elas que especificam as autorizações e restrições a serem impostas pelo mecanismo de controle de acesso.

- **Modelo de Segurança:** Fornece uma representação formal que descreve políticas de segurança e o seu funcionamento. Esta formalização permite, dentre outras coisas, a prova de propriedades desejadas para o sistema implementado.

- **Mecanismo de Segurança:** Define as funções de baixo nível que permitirão implementar o controle imposto pelas políticas e formalmente especificado pelo modelo.

Esta abordagem, além de possibilitar a separação entre os requisitos de proteção e o mecanismo que implementará a proteção, permite que políticas e mecanismos de segurança sejam comparados mais facilmente entre si e torna possível a verificação e validação de um sistema tanto com relação as políticas de segurança a serem implementadas quanto ao próprio mecanismo (Samarati e Vimercati, 2001).

5.5 Modelos de Controle de Acesso

Uma política de segurança deve capturar todas as regulamentações necessárias para garantir a integridade e sigilo de dados e recursos computacionais. Entretanto, para evitar interpretações ambíguas, idealmente estas políticas devem ser representadas segundo um modelo de segurança, ou modelo de controle de acesso, baseado em notação formal (Samarati e Vimercati, 2001).

Normalmente, os modelos de controle de acesso são agrupados em três categorias: Controle de Acesso Discricionário, Controle de Acesso Mandatório e Controle de Acesso Baseado em Papeis. Nas seções a seguir cada um destes modelos será discutido em maiores detalhes.

5.5.1 Controle de Acesso Discricionário

O modelo de Controle de Acesso Discricionário (do inglês *Discretionary Access Control* - DAC) é um paradigma que restringe controle de acesso a objetos baseado na identidade de sujeitos e/ou grupos no qual eles podem pertence. O controle é dito discricionário pois um sujeito detentor de uma permissão de acesso pode repassá-la para qualquer outro sujeito, direta ou indiretamente. Um exemplo de mecanismo baseado em DAC são as listas de controle de acesso usadas em ambientes UNIX (Li, 2011; Samarati e Vimercati, 2001).

5.5.2 Controle de Acesso Mandatório

O modelo de Controle de Acesso Mandatório (do inglês *Mandatory Access Control* - MAC) realiza o controle de acesso com base em regulações delegadas por uma autoridade central seguindo relações de hieraquia (Upadhyaya, 2011). Frequentemente utilizada no meio militar, a forma mais comum de MAC é a política de segurança multinível. Uma política de segurança multinível associa objetos e usuários de um sistema a classes de acesso organizadas segundo relações de dominância (Samarati e Vimercati, 2001). As categorias mais comuns são nível de segurança e conjunto de categorias. Políticas MAC

podem ter relações de dominância entre classes representadas graficamente usando uma estrutura denominada reticulado de segurança (Figura 5.2).

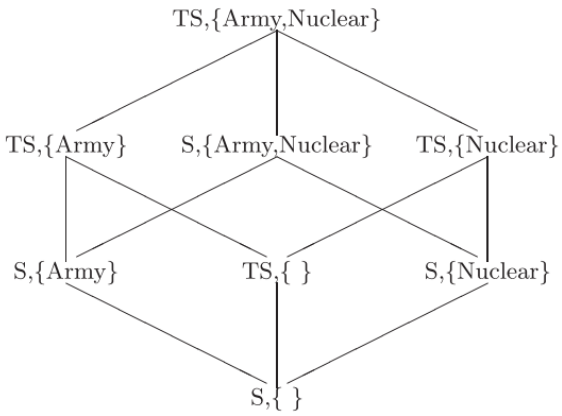


Figura 5.2: Exemplo de Reticulado de Segurança (Samarati e Vimercati, 2001)

Nível de segurança é um elemento de um conjunto parcialmente ordenado, ou seja, seus elementos organizam-se segundo relações de dominância ($x \succeq y$, lê-se x domina y) (Samarati e Vimercati, 2001). Categoria é um elemento de um conjunto que pode refletir, por exemplo, as áreas funcionais ou competências de uma ou mais organizações. Um exemplo de política de segurança MAC pode ser visto na figura 5.2 onde os níveis Ultra-Secreto (TS), Secreto (S), Confidêncial (C) e Não classificado (U), em que $TS \succeq S \succeq C \succeq U$, estão organizadas junto das categorias {Army, Nuclear}. Usando o modelo de controle de acesso MAC, permissões podem ser ativadas considerando as categorias superiores ou inferiores a um determinado nível.

5.5.3 Controle de Acesso Baseado em Papeis

O modelo de Controle de Acesso Baseado em Papeis (do inglês, *Role-Based Access Control* - RBAC) é considerado a mais importante inovação na gerência de acesso e identificação desde o controle de acesso discricionário e mandatório (Anderson, 2008). RBAC se baseia no conceito de agrupamento de privilégios (Samarati e Vimercati, 2001) para reduzir a complexidade de tarefas de administrativas de segurança ligadas à atribuição e revisão de permissões a usuários (Alturi e Ferraiolo, 2011), permitindo mapear mais naturalmente políticas de segurança a uma estrutura organizacional.

Em sistemas RBAC, um papel consiste de um agente organizacional detentor de um conjunto responsabilidades e atividades em uma organização. A partir desse conceito, papéis podem ser definidos e utilizados para intermediar o processo de atribuição e revogação de permissões aos usuários de um sistema, facilitando a gerência de segurança

(Elliott e Knight, 2010). Dois cenários ilustrando o impacto do uso do modelo RBAC pode ser visualizado na Figura 5.3.

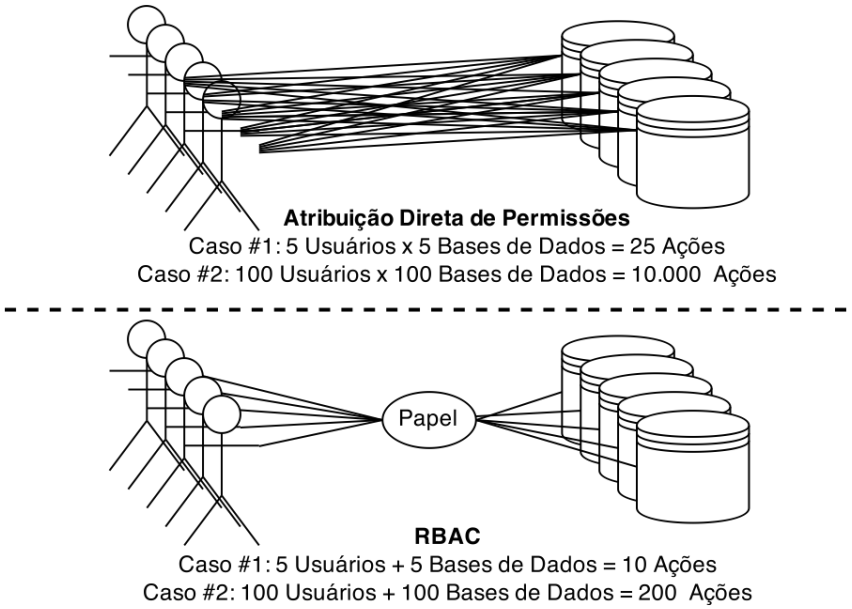


Figura 5.3: Impacto do RBAC na Gerência de Segurança

Considerando um cenário onde permissões são atribuídas diretamente aos usuários com cinco usuários e cinco bases de dados (Caso #1), o total de ações administrativas necessárias para habilitar o acesso de todos os usuários a todas tabelas é igual a 25 (5×5). Caso o número de usuários e de bases de dados sejam ambos 100 (Caso #2), o número de ações administrativas cresce para os 10.000 (100×100). Utilizando RBAC, por sua vez, o número de ações administrativas no Caso #1 é reduzido para 10 ($5 + 5$), enquanto que no Caso #2 o total de ações decai para 200 ($100 + 100$), caso seja considerado um único "Papel" compartilhado entre os usuários.

Vantagens do RBAC

A abordagem de Controle de Acesso Baseado em Papéis apresenta como principais vantagens: Gerência de Autorização, Hierarquia de Papéis, Privilégio Mínimo, Segregação de Funções e Aplicação de Restrições (Samarati e Vimercati, 2001). A seguir cada uma delas será discutida.

- **Gerência de Autorização:** O principal benefício do RBAC reside na independência lógica entre as atividades de atribuição de papeis a usuários e de atribuição de permissões a papéis. Este princípio simplifica a gerência de segurança, como pode ser visto no cenário ilustrado na Figura 5.3.

- **Hierarquia de Papéis:** É comum em organizações a existência de cargos ou funções que especializam ou generalizam uns aos outros, como em uma relação de herança. Sistemas RBAC podem permitir que seus administradores definam hierarquias de papéis a fim de propagar permissões entre papéis.

- **Privilégio Mínimo:** O uso de papéis permite controlar com maior facilidade os privilégios atribuídos a um usuários.

- **Segregação de Funções:** Do inglês *Separation of Duties* (SoD), é um princípio utilizado pra formular políticas de controle de acesso envolvendo múltiplas pessoas para a realização de um conjunto de tarefas. Dessa forma, fraudes tornam-se mais difíceis de serem realizadas devido o compartilhamento de responsabilidades entre vários participantes (Zurko e Simon, 2011).

- **Aplicação de Restrições:** Restrições de cardinalidade podem ser especificadas a fim de limitar o número de usuários habilitados a usar um dado papel ou o número máximo de papeis que podem usufruir de um privilégio específico.

Padrão NIST RBAC

Em resposta ao crescente interesse de organizações governamentais e da indústria em sistemas de RBAC, o Instituto Nacional de Padrões e Tecnologia dos Estados Unidos da América (do inglês *National Institute of Standards and Technology* - NIST) propos o padrão ANSI INCITS 359-2004. Este documento disponibiliza um modelo de referência e uma especificação funcional e administrativa para sistemas RBAC. Ele pode ser usado tanto por engenheiros de software interessados em projetar sistemas de controle de acesso como gerentes que queiram avaliar ou adquirir soluções de software RBAC (ANSI, 2004).

Modelo de Referência ANSI RBAC

O modelo de referência proposto pela ANSI (2004) define RBAC em termos de quatro componentes: *Core RBAC*, *Hierarchical RBAC*, *Static Separation of Duty Relations* e *Dynamic Separation of Duty Relations*. Cada componente será ilustrado e discutido em detalhes a seguir.

Core RBAC: Define o conjunto mínimo de elementos que devem ser disponibilizados por um sistema RBAC. Os elementos do RBAC são: usuários (USERS), papéis (ROLES), permissões (PRMS), operações (OPS) e objetos (OBS). Um sistema RBAC também deve contar com as funcionalidades de atribuição de relação entre usuário e papel (do inglês, *User Assignment* - UA) e entre papel e permissão (do inglês, *Permission Assignment* - PA). Uma permissão (PRMS) define um conjunto de operações (OPS) possíveis de serem

aplicadas sobre objetos (OBS). O *Core RBAC* também define o conceito de ativação de relações, que pode ser usado para gerenciar seções (SESSIONS) de usuários (*user_session* e *session_roles*). Na Figura 5.4 pode ser visualizado um esquema descrevendo o componente *Core RBAC*. Os demais componentes citados a seguir podem ser implementados opcionalmente.

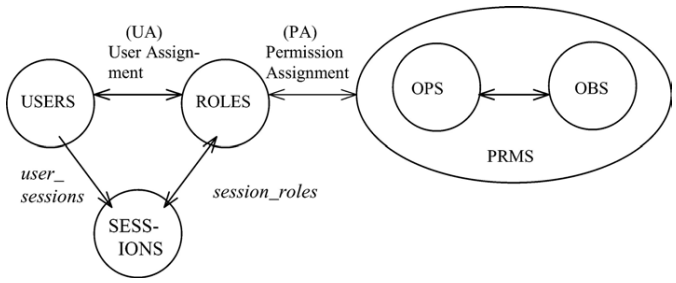


Figura 5.4: Core RBAC - (ANSI, 2004)

Hierarchical RBAC: Este componente, além dos elementos existentes no *Core RBAC*, define o conceito de relações hierárquicas entre papéis (do inglês, *Role Hierarchy* - RH). Matematicamente uma relação hierárquica consiste de um conjunto parcialmente ordenado definindo relações de superioridade entre papéis. Papéis de nível superior adquirem as permissões daqueles de nível inferior. Na Figura 5.5 pode ser visualizado um esquema descrevendo o componente *Hierarchical RBAC*.

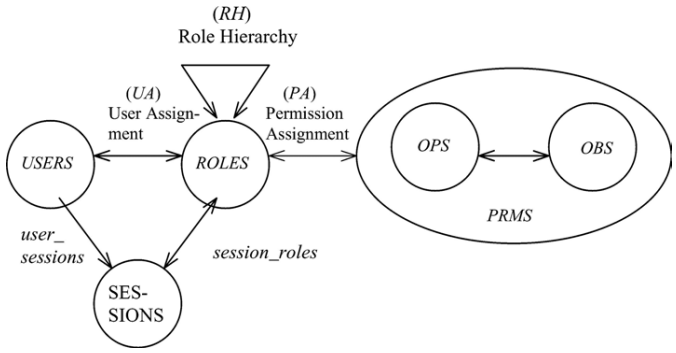


Figura 5.5: Hierarchical RBAC - (ANSI, 2004)

Static Separation of Duty Relations: Em sistemas baseados em papéis, é comum que administradores enfrentem situações onde existam conflitos de interesses entre papéis (e.g., um mesmo usuário sendo Caixa e Supervisor de uma operação). Um meio de prevenir esse tipo de conflito é por meio da definição de restrições de segregação de funções (do inglês, *Separation of Duty* - SoD). Restrições de SoD que permitem restringir a atribuição de papéis a usuários são denominadas restrições estáticas de SoD (do inglês, *Static Separation of Duty* - SSoD ou SSD).

Basicamente, em restrições do tipo SSoD são definidos um conjunto de papéis e um número máximo de atribuições que delimita o total de papéis desse conjunto no qual um

usuário pode ser atribuído. Por exemplo, a restrição SSD ({Caixa, Supervisor}, 2) pode ser entendida como uma regra impedindo usuários de serem atribuídos simultaneamente aos papéis Caixa e Supervisor. Restrições SSD podem ser estendidas para relações hierárquicas entre papéis. Na Figura 5.6 pode ser visualizado um esquema descrevendo o componente *SSoD RBAC*.

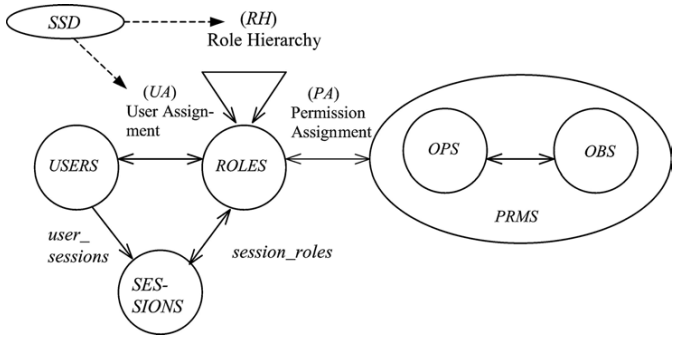


Figura 5.6: Static SoD - (ANSI, 2004)

Dynamic Separation of Duty Relations: Além de restrições de segregação de funções estáticas, o ANSI RBAC também conta com o componente de segregação de funções dinâmicas (do inglês, *Dynamic Separation of Duty* - DSoD ou DSD). Restrições do tipo DSoD, da mesma forma como SSoD, também têm o intuito de limitar permissões aos quais um usuário tem acesso. Entretanto, DSoD difere do SSoD por ser voltado para a ativação de relações (seções) ao invés da atribuição de papéis. DSoD permite autorizar usuários a ter acesso a dois ou mais papéis evitando conflitos de interesse, quando atuando separadamente. Por exemplo, em um sistema com DSoD um usuário pode estar atribuído aos papéis Caixa e Supervisor, entretanto restrições dinâmicas podem impedir esse mesmo usuário de atuar com ambos os papéis durante uma única seção. Caso o usuário deseje atuar como Caixa ele precisará abandonar a seção usando o papel Supervisor, ou vice versa, evitando a ocorrência de conflitos de interesse. Na Figura 5.7 pode ser visualizado um esquema descrevendo o componente *DSoD RBAC*.

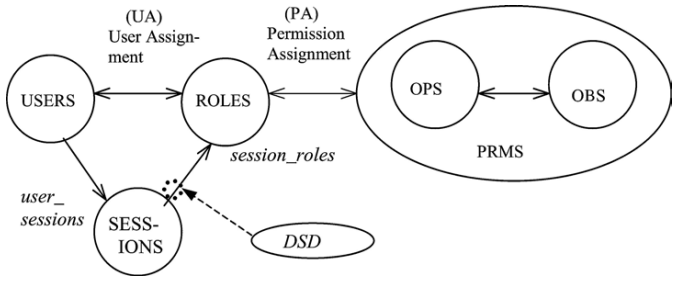


Figura 5.7: Dynamic SoD - (ANSI, 2004)

Especificação Funcional e Administrativa para Sistemas RBAC

A especificação funcional e administrativa do Padrão ANSI RBAC descreve o conjunto de *operações administrativas*, *funções de suporte de sistema* e *funções de revisão administrativa* requeridas em sistemas RBAC (ANSI, 2004). Em ANSI (2004) essas funções encontram-se formalmente especificadas segundo um subconjunto da notação Z a fim de favorecer a realização de testes de conformidade. A seguir serão citados alguns dos requisitos funcionais e administrativos especificados no padrão ANSI RBAC (ANSI, 2004).

Operações Administrativas: As operações administrativas do modelo de referência dizem respeito a tarefas como adição e remoção de usuários, adição e remoção de papéis, atribuição e desatribuição de papéis, adição ou deleção de herança, criação, adição e deleção de SoD estáticos e dinâmicos, concessão e revocação de permissões.

Funções de Suporte de Sistema: As operações de suporte de sistema especificam operações como criação e deleção de sessões, ativação e desativação de papéis e checagem de acesso.

Funções de Revisão Administrativa: Funções de revisão administrativa são formadas por operações relacionadas a consulta de usuários, papéis, permissões, relações usuário-papel, papel-permissão e sessões existentes ou em uso.

Exemplo de Política RBAC

A seguir, na Equação 5.1, será ilustrado um exemplo de política RBAC no contexto de um sistema de submissão e revisão de artigos.

$$\begin{aligned}
\text{USERS} &= \{Alice, Bob, Tales\} = \{u1, u2, u3\} \\
\text{ROLES} &= \{Autor, Revisor\} = \{r1, r2\} \\
\text{PRMS} &= \{SubmeterArtigo, LerParecer, RevisarArtigo\} \\
&= \{p1, p2, p3\} \\
\text{UA} &= \{(u1, r1), (u1, r2), (u2, r1), (u3, r2)\} \\
\text{PA} &= \{(p1, r1), (p2, r1), (p3, r2)\} \\
DSoD &= (\{r1, r2\},\ 2)
\end{aligned} \tag{5.1}$$

Na Equação 5.1 pode ser visto um o exemplo de política onde três usuários – *Alice*, *Bob* e *Tales*, estão associados aos seus respectivos papeis. *Alice* está atribuído aos papéis *Autor* e *Revisor*, *Bob* ao papel *Autor* e *Tales* ao papel *Revisor*. O papel *Autor* possui as permissões *SubmeterArtigo* e *LerParecer* e o papel *Revisor* tem a permissão *RevisarArtigo*. Além disso, a fim de evitar conflito de interesses, também foi definida uma restrição dinâmica de segregação de funções (DSoD) impedindo que usuários ativem ambos os papéis *Autor* e *Revisor* durante uma

seção. A restrição $DSoD = (\{r1, r2\},\ 2)$ impede que um usuário atue simultaneamente usando dois ou mais papéis do grupo $\{r1, r2\}$. Essa restrição impede, por exemplo, que o usuário *Alice* que está atribuído aos papéis *Autor* e *Revisor* consiga revisar um artigo no qual ele é autor.

5.6 Softwares de Controle de Acesso

Atualmente, RBAC tem sido implementado para diversos fins, como gerenciamento de sistemas operacionais, de banco de dados, de infraestrutura de redes, ambientes colaborativos e virtualizados (Alturi e Ferraiolo, 2011). A seguir, serão citados alguns exemplos de ferramentas de controle de acesso que implementam parcial ou integralmente o modelo ANSI RBAC.

X-GTRBAC

X-GTRBAC é um framework Java e linguagem de especificação de políticas RBAC baseada em XML propostos por Bhatti et al. (2005). X-GTRBAC consiste basicamente de dois módulos (Figura 5.8): um inicializador de políticas (*XML Processor*) e um realizador de políticas (*GTRBAC Processor*). O inicializador carrega políticas RBAC especificadas em formato XML enquanto que o realizador garante ou nega acesso com base nas relações usuário-papel ativas.

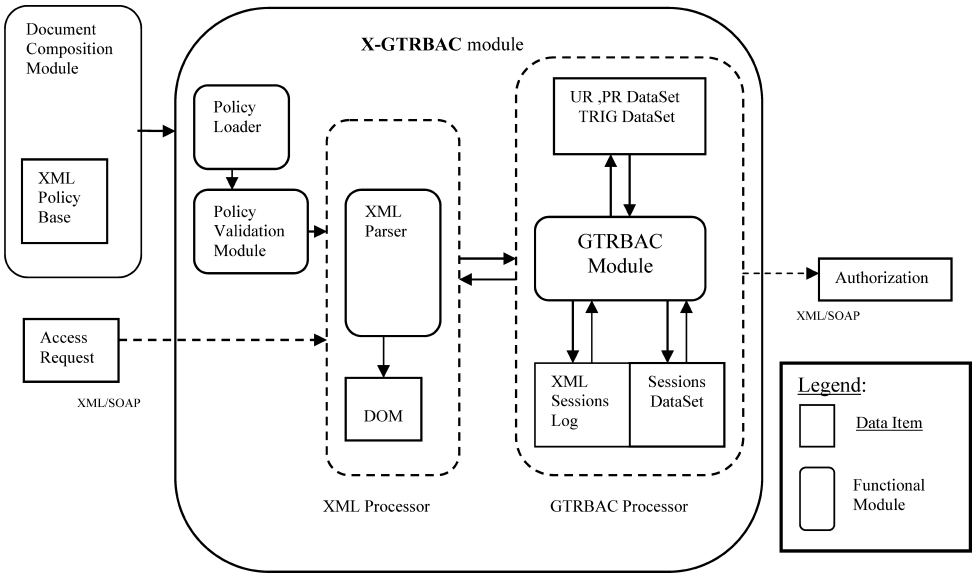


Figura 5.8: X-GTRBAC - Arqutetura de Sistema (Bhatti et al., 2005)

O módulo *XML processor* é implementado em linguagem Java usando a biblioteca *Java API for XML Processing* (JAXP). Módulos foram projetados para realizar a leitura e instanciação dos documentos XML e repassá-los ao módulo *GTRBAC processor*. O módulo *GTRBAC processor* é o responsável por adminstrar e fazer cumprir as políticas especificadas em XML (*XML Policy Base*). Com esta infraestrutura, requisições enviadas ao mecanismo são analisadas e um parecer de autorização é emitido com base nas políticas.

PERMIS

PERMIS é um sistemas distribuído de autorização baseado em políticas RBAC proposto por Chadwick e Otenko (2003). Ele fornece uma infraestrutura para gerenciamento de privilégio, gerenciamento de confiança e tomada de decisão. As políticas de segurança são definidas na forma de subpolíticas que especificam usuários, papéis, ações, objetivos e as permissões alocadas aos papeis. PERMIS tem sido usado como mecanismo de autenticação em diversos domínios, tais como sistemas baseados em grid computacional e nos mecanismos de autenticação Shibollet e Kerberos (Colombo et al., 2010).

Apache Fortress

Apache Fortress, ou simplesmente Fortress, é um Kit de Desenvolvimento de Software Java de código aberto para gerenciamento de identidade de acesso para sistemas em conformidade com a tecnologia *Lightweight Directory Access Protocol* (LDAP) versão 3, usada para prover acesso a diretórios via rede IP. Fortress é 100% compatível com o padrão ANSI RBAC e compõe a arquitetura de projetos de código aberto de grande porte como OpenLDAP e Apache Tomcat (Apache, 2014a).

OWASP PHP-RBAC

PHP-RBAC é uma biblioteca PHP de controle de acesso RBAC compatível com o nível 2 (*Hierachical RBAC*) do padrão ANSI RBAC. Ela fornece uma infraestrutura para gerenciamento de controle de acesso que pode ser utilizada em aplicações de pequeno porte ou de escala corporativa (OWASP, 2014b). É um projeto de código aberto que tem sido testado em contexto industrial por mais de três anos e que recebe incentivo da *Open Web Application Security Project* (OWASP), uma organização sem fins lucrativos de reconhecimento internacional voltada para o estudo e melhoria da segurança de software (OWASP, 2014a).

Apache Redback

Apache Redback é um projeto código aberto da Fundação Apache que disponibiliza uma série de componentes de software direcionados a segurança de aplicações web. Ele é composto por uma série de módulos voltados para a realização de tarefas ligadas a autenticação, autorização e gerenciamento de usuários. Ele é utilizado como componente responsável pela segurança e controle de acesso em outros projetos da Fundação Apache, como o sistema de integração contínua Apache Continuum e Archiva (Apache, 2014b).

grSecurity

grSecurity é um complemento para o kernel do Linux voltado para garantia de segurança em nível de sistema operacional. Ele fornece uma série de funcionalidades para detecção, prevenção e contenção de acesso a recursos de sistema utilizando o modelo de controle de acesso baseado em papéis, complementando o controle de acesso discricionário nativamente existente no Linux. Dentre as funcionalidades disponibilizadas, ele ajuda na proteção contra ameaças de elevação de privilégio, execução de código malicioso, corrupção de memória, além de permitir a realização de auditorias no sistema operacional (Bugliesi et al., 2012; Open Source Security, 2014).

5.7 Considerações Finais

Neste capítulo, foram discutidos conceitos básicos sobre controle de acesso. Os três principais modelos de controle de acesso (MAC, DAC e RBAC) foram definidos e ilustrados com exemplos. O modelo RBAC foi apresentado em mais detalhes devido ser o domínio de aplicação desse trabalho de mestrado. Os elementos do modelo RBAC (usuários, papéis, permissões e restrições) foram apresentados junto com os benefícios que o eu uso pode proporcionar. O padrão ANSI RBAC com seus quatro componentes também foi detalhado. Exemplos ilustrando cada um dos componentes foram apresentados a fim de facilitar o entendimento do que vem a ser uma política RBAC e os seus elementos. Também foram apresentados alguns exemplos de ferramentas para controle de acesso de código-aberto. Os conceitos e exemplos de aplicação apresentados neste capítulo podem ser usados para realizar estudos de carater prático em controle de acesso, tal como o experimento proposto para esse trabalho de mestrado.

Capítulo

6

Teste de Controle de Acesso Baseado em Modelos

6.1 Considerações Iniciais

Métodos formais podem apoiar inúmeras tarefas durante o projeto, verificação e implementação de mecanismos de segurança, tais como sistemas de controle de acesso (Huth, 2011). As atividades de análise e teste de políticas, por exemplo, têm sido domínio de aplicação para muitas técnicas formais de V&V, como *model checking* (Mondal et al., 2011) e teste de conformidade usando MEFs (Masood et al., 2009, 2010b). Em teste de segurança baseado em modelos (TSBM), diferentes tipos de modelos de especificação têm sido utilizados para descrever sistemas e automatizar a geração de casos de testes (Damasceno et al., 2014). Entretanto, TSBM ainda apresenta uma série de desafios e questões em aberto, tais como o a escalabilidades dos seus métodos (Utting et al., 2012).

Neste capítulo serão discutidos conceitos e definições relacionadas a análise e teste de políticas de controle de acesso usando modelos. Em especial, será dada ênfase ao teste de conformidade de sistemas RBAC usando máquinas de estados finitos pois, estudos preliminares apontaram uma carência de trabalhos neste domínio.

Na seção 6.2 será discutida a estratégia de modelagem usada para descrever sistemas de controle de acesso como sistemas de transição de estados. Na seção 6.3 serão discutidos conceitos básicos sobre análise de políticas. Ne seção 6.4 será discutido teste de políticas. Na seção 6.5 serão apresentados alguns trabalhos relacionados a teste de segurança baseado em modelos e

teste de controle de acesso usando modelos baseados em estados encontrados por meio de uma revisão e um mapeamento, respectivamente. Na seção 6.6 será discutida uma técnica, proposta por Masood et al. (2009), para teste de conformidade para sistemas de RBAC usando máquinas de estados.

6.2 Modelagem de Controle de Acesso

Segundo Li e Tripunitara (2006), um sistema de controle de acesso pode ser modelado como um sistema de transição de estados $< \Gamma, Q, \vdash, \Psi >$. Nesta quádrupla, Γ corresponde ao conjunto de estados do sistema, Q ao conjunto de consultas que podem ser enviadas ao sistema, Ψ é o conjunto de regras de mudança de estados e $\vdash$: $\Gamma \times Q \rightarrow \{true, false\}$ é denominado relação de implicação.

Um estado, $\gamma \in \Gamma$, contém todas as informações necessárias para realizar decisões de controle de acesso. Quando uma consulta, $q \in Q$, é recebida, uma resposta $\gamma \vdash q$ significa que o acesso necessário para a consulta q será garantido no estado γ, caso contrário $\gamma \nvdash q$, ou seja, ele será negado.

Uma regra de mudança de estado, $\psi \in \Psi$, determina como o sistema de controle de acesso muda de estado. Considerando dois estados $\gamma_1, \gamma_2 \in \Gamma$ e uma regra de mudança de estados $\psi \in \Psi$, dizemos $\gamma_1 \longmapsto_\psi \gamma_2$ se há uma regra que permite a mudança do estado γ_1 para o estado γ_2. Dizemos $\gamma_1 \overset{*}{\mapsto}_\psi \gamma_2$ se zero ou mais regras de mudanças de estados permitem sair do estado γ_1 para γ_2, ou seja, dizemos que γ_2 é Ψ-*Alcançável* a partir de γ_1.

Assim como será visto a seguir, esses conceitos podem ser usados como base tanto para a análise de propriedades de segurança como para o teste mecanismos de controle de acesso.

6.3 Análise de Políticas

Análise de políticas, ou análise de segurança, é uma atividade importante que permite identificar conflitos, lacunas e problemas de segurança em políticas, além de possibilitar o seu refinamento (Huth, 2011).

Formalmente, segundo Li e Tripunitara (2006), análise de política pode ser descrito como uma quádrupla $< \gamma, q, \vdash, \Pi >$ onde $\gamma \in \Gamma$ é um estado, $q \in Q$ é uma consulta, $\psi \in \Psi$ é uma regra de mudança de estado e $\Pi \in \{\exists, \forall\}$ é um quantificador. Dependendo do quantificador usado, uma análise fornecerá um tipo diferente de parecer sobre as propriedades de segurança do sistema modelado.

Uma instância de análise $< \gamma, q, \vdash, \exists >$ é uma análise que visa perguntar se existe um estado $\gamma_1 \in \Gamma$ tal que $\gamma \overset{*}{\mapsto}_\psi \gamma_1$ e $\gamma_1 \vdash q_x$. Caso a resposta para a instância da análise seja afirmativa, dizemos que q é possível dado γ e ψ.

Uma instância $< \gamma, q, \vdash, \forall >$ é uma análise que visa perguntar se, considerando todos os estados $\gamma_1 \in \Gamma$, podemos dizer que $\gamma \overset{*}{\mapsto}_{\psi} \gamma_1$ e $\gamma_1 \vdash q$. Caso a resposta para a instância da análise seja afirmativa, dizemos que q é necessário dado γ e ψ.

A partir destes conceitos, diversas perguntas podem ser efetuadas a fim de validar propriedades de segurança. Li e Tripunitara (2006) lista um conjunto de propriedades de segurança com exemplos de perguntas:

- **Segurança Simples (*Simple Safety*):** Existe algum estado alcançável no qual um usuário potencialmente não confiável pode ter acesso a um recurso? Uma resposta negativa, neste caso, indica que o sistema é confiável.
- **Disponibilidade Simples (*Simple Availability*):** Considerando todos os estados de um sistema, um conjunto de recursos estará sempre acessível a um usuário confiável? Uma resposta positiva indica que o sistema apresenta disponibilidade.
- **Segurança Delimitada (*Bounded Safety*):** Considerando todos os estados de um sistema, um conjunto de recursos têm seu uso limitado a um contexto específico? "Exclusão mútua" é um caso especial desta pergunta aplicável em casos onde há, por exemplo, restrições de segregação de funções (SoD) a serem testadas.
- **Vivacidade (*Liveness*):** Um conjunto de recursos, em algum dado momento, pode não ter usuários com permissão de acesso habilitada? Uma reposta negativa significa que um recurso sempre estará sob controle de alguém.
- **Contenção (*Containment*):** Um usuário u atribuído a um papel r tem acesso a permissão p, considerando que r está relacionado a p? Uma resposta positiva significa que a propriedade é válida.

Por meio da análise de segurança, requisitos de segurança podem ser especificados e a validade de propriedades de segurança pode ser comprovada.

6.4 Teste de Políticas

Teste de software é uma técnica importante que ajuda na detecção de defeitos e inconsistências em sistemas. Da mesma forma, defeitos podem ser detectadas na implementação de políticas por meio de testes (Figura 6.1).

No teste de políticas, os dados de entrada são equivalentes as requisições enviadas para o mecanismo de controle de acesso em teste (do inglês, *Access Control Under Test* - ACUT). Ao receber estas entradas, o ACUT retorna respostas (dados de saída) que podem ser comparadas com as respostas esperada (saídas esperadas). Dessa forma, discrepâncias entre o comportamento do ACUT e as políticas especificadas podem ser identificadas (Martin e Xie, 2007).

Assim como no teste de software tradicional, o teste de políticas também conta com uma série de métricas de cobertura e seleção de casos de teste específicas. Martin et al. (2006) e Hu

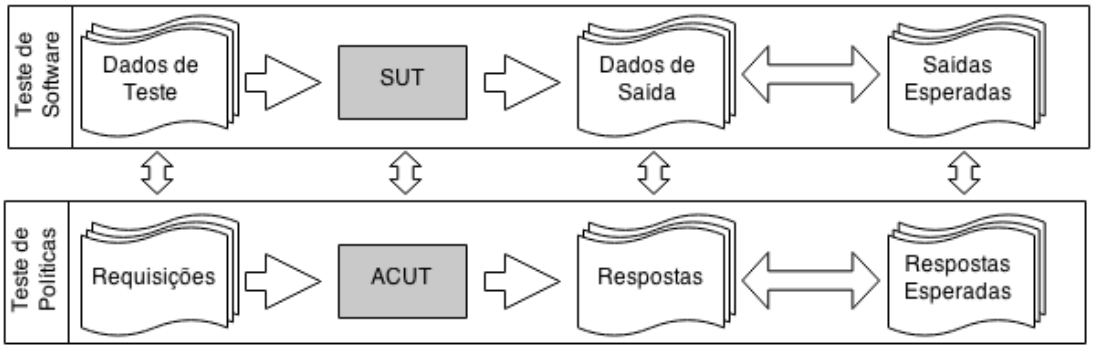


Figura 6.1: Teste de Software e Teste de Políticas

et al. (2007) propõem métricas para análise de cobertura para teste de políticas de controle de acesso especificadas em *eXtensible Access Control Markup Language* (XACML) usando análise de mutantes e e a cobertura elementos da política em XACML, como *rules* e *conditions*. Traon et al. (2007) discute uma abordagem para geração de casos de teste para políticas de controle de acesso e requisitos funcionais de sistemas web também usando mutação. Estas técnicas podem ser usadas para atestar a conformidade de um ACUT com os seus requisitos.

Teste de conformidade é um caso especial de teste baseado em modelos, que visa checar se um sistema em teste cumpre sua especificação segundo uma relação definida entre uma implementacão e/ou modelos (Bertolino, 2007). Por meio de modelos formais, tais como máquinas de estados e lógicas de predicado, sistemas de controle de acesso podem ser testados com base em especificações ou padrões.

Hansen e Oleshchuk (2005) propõem uma técnica de teste de conformidade entre políticas de controle de acesso RBAC e implementações baseada em *model checking*. Baumgrass et al. (2012) discutem uma técnica de teste de conformidade de políticas RBAC para sistemas de modelagem de processos baseada em lógica temporal linear. Enquanto que Power et al. (2011) utilizam a linguagem de modelagem Alloy para checar a conformidade entre políticas de controle de acesso e propriedades de segurança.

6.5 Revisão de Literatura

Motivado nessa variedade de abordagens de teste de controle de acesso, foram realizadas uma revisão e um mapeamento sistemáticos no contexto de teste de segurança baseado em modelos e teste de políticas RBAC com modelos baseado em estados. A seguir, serão discutidos os objetivos e resultados obtidos de cada um dos trabalhos.

6.5.1 Teste de Segurança Baseado em Modelos

A popularização de serviços baseados em mobilidade, virtualização e conectividade têm favorecido tanto a flexibilização de negócios de empresas quanto o aumento de vulnerabilidades aos

quais estas organizações estão se expondo (Rashid et al., 2013). Consequentemente, a demanda por técnicas avançadas de teste, verificação e validação, tal como o Teste de Segurança Baseado em Modelos (TSBM), tem crescido significativamente.

Neste contexto, Damasceno et al. (2014) realizaram uma revisão sistemática a fim de coletar evidências que fornecessem uma visão geral das pesquisas no contexto de TSBM. Para esse fim, as seguintes questões de pesquisa foram definidas:

> **RQ1:** Que resultados o uso de TSBM em serviços ligados a mobilidade, virtualização ou conectividade tem proporcionado?
>
> **RQ2:** Quais as características predominantes nos estudos de TSBM?
>
> **RQ2.1:** Considerando a taxonomia do TBM (Utting e Legeard, 2007), como as abordagens de TSBM podem ser categorizadas?
>
> **RQ2.2:** Que tipo de SUTs estas abordagens utilizam em seus estudos empíricos?
>
> **RQ2.3:** Quais as desvantagens e limitações destes estudos?
>
> **RQ2.4:** Estes estudos podem ser aplicados no tratamento de exfiltração de dados?

Ao todo, 23 artigos foram selecionados e lidos a fim de extrair informações que evidenciassem sua relação com a taxonomia do TBM e domínios de aplicação. As seguintes informações foram extraídas: título do artigo, base de origem, ano de publicação, informações correspondentes a cada dimensão da taxonomia TBM, o tipo de ameaça modelada, o tipo de SUT testado, a relação com mobilidade, virtualização, conectividade e exfiltração de dados, vantagens, desvantagens e limitações. As conclusões obtidas a partir dos dados extraídos são apresentadas a seguir:

RQ1: Que resultados o uso de TSBM em serviços ligados a mobilidade, virtualização ou conectividade tem proporcionado?

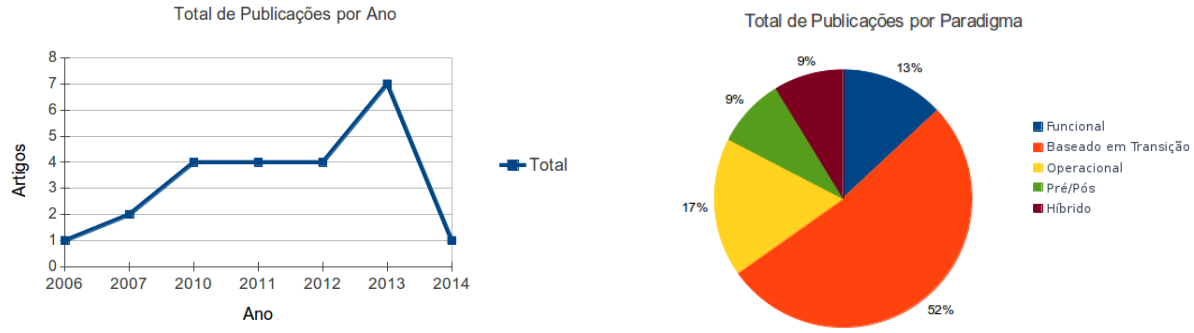
Com relação a **RQ1**, foi identificado que o TSBM facilitou, dentre outras tarefas, a especificação de sistemas e do comportamento de atacantes, reduziu a ambiguidade dos planos de testes e facilitou a geração e replicação de testes.

Outro ponto identificado foi que praticamente todos os estudos permitiam a geração de dados de entradas com oráculos de teste. Isso mostra o grau de automação que abordagens de TBM para segurança pode proporcionar. Somente seis estudos identificados aplicaram TBM no teste de serviços com requisitos relacionados à mobilidade, virtualização ou conectividade.

RQ2: Quais as características predominantes nos estudos de TSBM?

Com relação a **RQ2**, foi constatada uma tendência crescente na quantidade de publicações ao longo dos anos e uma predominância de trabalhos usando modelos baseados em transição, como máquinas de estados finitos. Artigos usando mais de uma notação foram categorizados como

Híbrído. Os gráficos com o Total de Publicações por Ano e Total de Publicações por Paradigma podem ser vistos respectivamente nas Figuras 6.2(a) e 6.2(b).



(a) Publicações por Ano **(b)** Publicações por Paradigma

Figura 6.2: Total de Publicações

RQ2.1: Considerando a taxonomia do TBM, como as abordagens de TSBM podem ser categorizadas?

A questão **RQ2.1** mostrou que nem todos os estudos apresentam claramente informações que permitam classificá-los segundo todas as dimensões da taxonomia do TBM, tais como critério de seleção e tecnologia de geração de testes. Além da dimensão paradigma, foi possível extrair o assunto modelado, sendo a maioria voltado para a modelagem exclusivamente do SUT. Com relação a execução dos testes, foi percebida uma predominância de abordagens de teste *offline*. Um ponto a ser ressaltado é que o fato destes estudos não terem algumas dimensões claramente descritas pode dificultar a realização de estudos sistemáticos.

RQ2.2: Que tipo de SUTs estas abordagens utilizam em seus estudos empíricos?

No contexto da **RQ2.2**, foi identificada uma predominância de 74% de estudos experimentais usando sistemas reais ou de grande porte em diferentes domínios. Alguns dos domínios encontrados foram: sistemas para *smart-cards*, softwares da área de saúde e políticas de segurança.

RQ2.3: Quais as desvantagens e limitações destes estudos?

Quanto a **RQ2.3**, foi constatado que quatro abordagens não cobriam o processo de TBM por completo. O estudo de Ouerdi et al. (2013), por exemplo, não discute geração de casos de teste executáveis. Bozic e Wotawa (2013) e Lebeau et al. (2013) citam a ausência de reuso de modelos como uma limitação da sua abordagem. E Xu et al. (2012) cita o esforço necessário para realizar a modelagem do SUT e o comportamento de atacantes como uma desvantagem.

RQ2.4: Estes estudos podem ser aplicados no tratamento de exfiltração de dados?

Quanto a **RQ2.4**, nenhum estudo tratou especificamente sobre exfiltração de dados. Entretanto, haviam trabalhos discutindo ameaças relacionadas a exfiltração de dados quebra de sigilo de dados e de requisitos de confidencialidade. Isso sugere que o TBM talvez possa ser usado para indiretamente mitigar ou detectar esse tipo de ameaça.

6.5.2 Teste de Controle de Acesso Baseado em Estados

Após a revisão em teste de segurança baseado em modelos ter sido realizada, um mapeamento sistemático foi executado a fim de obter uma visão mais detalhada do perfil das pesquisas em teste de controle de acesso usando notações baseadas em estados. O fato deste mapeamento ter como foco notações baseadas em estados se justifica nas evidências de Damasceno et al. (2014) que apontam esse paradigma como predominante no contexto de TSBM. O domínio de aplicação de controle de acesso foi escolhido devido ele ser um dos principais mecanismos de mitigação de ameaças (Samarati e Vimercati, 2001).

A partir disso, um mapeamento foi realizado a fim de identificar as notações baseadas em estados que têm sido usadas para testar políticas e sistemas de controle de acesso. A abordagem PICO (***P****opulation*, ***I****ntervention*, ***C****omparison*, ***O****utcome*) (Kitchenham e Charters, 2007) foi usada para definir as questões de pesquisa deste mapeamento. A população (*Population*) ficou estabelecida como "Sistemas e Políticas de Controle de Acesso", intervenção (*Intervention*) como "Notações Baseadas em Estado" e resultado (*Outcome*) como "Teste de Software". O elemento comparação (*Comparison*) não foi definido pois não havia interesse em comparar estudos. As seguintes questões de pesquisa (RQ) foram estabelecidas:

RQ1: Quais são as notações baseadas em estados que têm sido usadas para o teste de políticas de controle de acesso?

RQ1.1: A abordagem inclui oráculos de teste?

RQ1.2: Quais são os critérios de seleção de testes usados?

RQ1.3: Quais são as tecnologias de geração de teste usadas?

RQ1.4: Qual é a abordagem de execução de teste mais usada?

RQ2: Quais são os modelos de controle de acesso que têm sido testados usando notações baseadas em estados?

RQ2.1: Que tipos de problemas são tratados pela abordagem de teste?

RQ2.2: Que políticas são usadas como exemplo?

RQ2.3: Que programas são usados como SUT?

A questão de pesquisa RQ1 teve como objetivo identificar as notações baseadas em estados que têm sido usadas para especificar sistemas de controle de acesso e gerar testes. As questões RQ1.1, RQ1.2, RQ1.3 e RQ1.4 foram definidas a fim de fornecer uma visão geral do teste de controle de acesso usando modelos baseados em estados considerando, respectivamente, oráculos de teste, critérios de seleção, técnicas de geração e execução de testes. Para isso, foram inseridos na *string* de busca deste trabalho fragmentos da *string* do trabalho de Shafique e Labiche (2013) sobre ferramentas de teste baseado em estados.

A questão de pesquisa RQ2 teve como objetivo identificar os modelos de controle de acesso testados usando as notações identificadas com a RQ1. Para isso, termos relacionados a controle de acesso foram incluídos na *string* de busca. Com as subquestões RQ2.1, RQ2.2 e RQ2.3 esperou-se identificar, respectivamente, os problemas tratados nos artigos. Duas categorias de problemas foram consideradas: análise de segurança e teste de políticas. Para cobrir a RQ2 e subquestões, foram inseridos na *string* de busca os termos "teste" "verificação" e "validação".

A partir disso a seguinte string de busca foi definida: ("Access Control" OR "Control Policy" OR "Control Policies") AND ("transition system" "statechart" OR "statecharts" OR "state model" OR "state machine" OR "state machines" OR "automata") AND ("Testing" OR "Test" OR "Verification" OR "Validation"). O trabalho de Masood et al. (2010b) foi usado como artigo de controle para validar a *string* pois, em Damasceno et al. (2014) ele foi o único trabalho voltado para o teste de conformidade de sistemas de controle de acesso.

Quanto aos critérios de inclusão e exclusão, foram definidos que somente artigos discutindo teste de controle de acesso baseado em RBAC usando notações baseadas em estados e realizando validações experimentais seriam incluídos. Trabalhos que não estivessem escritos em inglês, não discutissem controle de acesso baseado em RBAC ou TBM usando modelos baseados em estados seriam excluídos. Resumos, editoriais e trabalhos que não tivessem como foco principal teste usando notações baseadas em estados de políticas RBAC também não foram incluídos.

Ao término da etapa de inclusão e exclusão, 10 artigos foram selecionados. Os artigos selecionados encontram-se listados na Tabela 6.1. Cada um dos artigos foi lido e os seguintes dados foram extraídos: Título, Autores, Ano, Veículo de publicação (Origem) e Tipo de publicação. As seguintes informações também foram extraídas, considerando a taxonomia do TBM (Utting et al., 2012): Notação de Modelagem usada, Escopo da Modelagem, Características do Modelo, Critério de Seleção, Tecnologia de Geração de Testes, Abordagem de Execução. Modelo de Controle de Acesso, Problema tratado, Política e ACUT testados também foram extraídos a fim de responder as questões de pesquisa previamente estabelecidas. As políticas em teste e os SUTs coletados foram coletados a fim de suprir quaisquer demandas futuras de artefatos em teste para trabalhos experimentais. A categorização dos trabalhos segundo problema tratado permitiria identificar possíveis lacunas ou tendências de pesquisa. Os dados extraídos encontram-se na Tabela 6.2 e são discutidos a seguir.

RQ1: Quais são as notações baseadas em estados que têm sido usadas para o teste de políticas de controle de acesso?

Dos artigos selecionados, quatro usam Autômatos Temporizados (TA) como notação de modelagem, um usa Autômato Temporizado com Entrada e Saída (TIOA), um usa Sistema de Transição Rotulado (LTS), dois usam Máquina de Estados Finitos Extendida (EFSM) e dois usam Máquina de Estados. Na Figura 6.3 pode ser visto um gráfico mostrando a quantidade de estudos usando cada uma das notações identificadas.

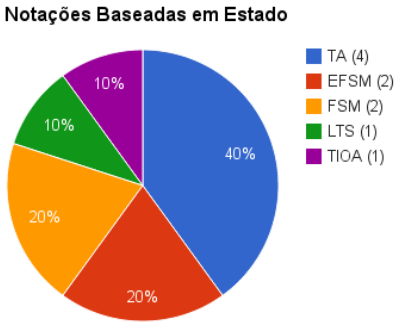


Figura 6.3: Notações Baseada em Estados Utilizadas

Vale ressaltar que todos os trabalhos, com excessão do artigo A2 (Song e Chen, 2012), utilizam as próprias políticas de controle de acesso para gerar os testes automaticamente, sem a necessidade de uma remodelagem do SUT/ACUT. Isso evidencia o potencial de automação das abordagens de TBM em controle de acesso, confirmando as evidências obtidas em (Damasceno et al., 2014).

Em controle de acesso, esta taxa de reuso pode ser justificada na natureza no desenvolvimento multi-fase de sistemas de controle de acesso. Enquanto o TBM não encoraja o reuso de modelos de projeto a fim de mitigar as chances de propagação de falhas de projeto (Utting et al., 2012) para o teste, o desenvolvimento multi-fase possibilita uma validação independente das políticas e mecanismos de segurança. Dessa forma, o teste do requisito de controle de acesso pode ser realizado em diferentes níveis, o que favorece o reuso de políticas como modelos para geração e análise de testes.

RQ1.1: A abordagem inclui oráculos de teste?

Como pode ser visto na Tabela 6.2 todos os estudos dispõem de um oráculo de teste. Uma justificativa para isso foi exposta na resposta para a RQ1 desta revisão.

RQ1.2: Quais são os critérios de seleção de testes usados?

O critério de seleção baseado em cobertura estrutural foi usado em todos os dez artigos. Além da cobertura estrutural, os artigos (Masood et al., 2009) e (Masood et al., 2010b) também usaram

critérios baseados em especificação e baseado em requisitos. Na Figura 6.4 pode ser visto um gráfico ilustrando o total de estudos por critério de seleção de testes.

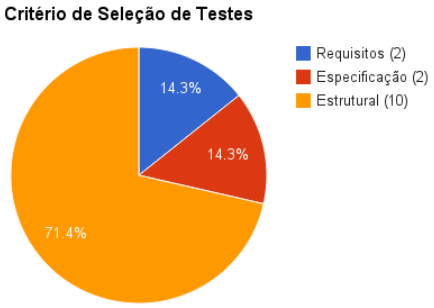


Figura 6.4: Critério de Seleção de Testes

Os dois artigos (Masood et al., 2009, 2010b) foram classificados com o critério de seleção baseado em requisitos, pois as heurísticas usaram requisitos funcionais do ACUT como referência. Alguns exemplos de heurísticas usadas são o total de requisições enviadas (usado para definir comprimento da sequência de testes aleatóriamente gerada), a combinação entre atribuição de papel e ativações possíveis e as ativações e atribuições para cada usuários/papéis. Os dois artigos usando critério de seleção baseado em especificação (Masood et al., 2009, 2010b) foram classificados dessa forma pois o tamanho das sequências de teste foi interpretado como uma especificação de requisito para seleção de casos de teste.

RQ1.3: Quais são as tecnologias de geração de teste usadas?

A tecnologia de geração de testes *Model Checking* foi classificada como a mais utilizada (6 artigos - 50%), seguido da geração de testes baseada em busca (4 artigos - 33%). Dois artigos utilizaram geração aleatória como tecnologia de geração de testes (17%). Na Figura 6.5 pode ser visto um gráfico ilustrando o total de estudos por tecnologia de geração de testes.

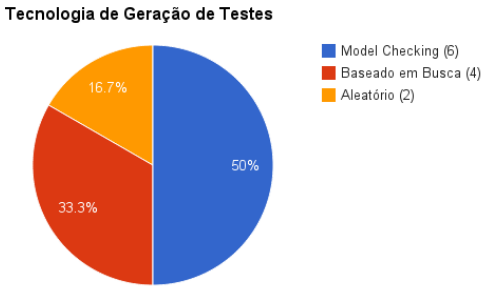


Figura 6.5: Tecnologia de Geração de Testes

Destes trabalhos, somente os artigos (Masood et al., 2009), (Masood et al., 2010b) e (Masood et al., 2010a) apresentaram explícitamente os algorítmos usados. Nos três artigos o Método W, apresentado na Seção 4.3.2.4, foi escolhido como tecnologia de geração de testes.

RQ1.4: Qual é a abordagem de execução de teste mais usada?

Todos os trabalhos propuseram técnicas *offline* de teste baseado em modelos.

RQ2: Quais são os modelos de controle de acesso que têm sido testados usando notações baseadas em estados?

Como o objetivo desta revisão ficou restrito ao modelo RBAC, já era esperado que os trabalhos ficassem restritos ao próprio RBAC e suas variantes. Dentre os trabalhos selecionados, foram identificados cinco voltados para o modelo RBAC (Bugliesi et al., 2012; Dury et al., 2007; Masood et al., 2009, 2010b; Song e Chen, 2012), dois para o RBAC Temporal (TRBAC) (Masood et al., 2010a; Mondal e Sural, 2008), dois para o RBAC Temporal Generalizado (GTRBAC) (Mondal et al., 2009, 2011) e um para o RBAC Espaço-Temporal (STRBAC) (Geepalla et al., 2012). Na Figura 6.6 pode ser visto um gráfico ilustrando o total de estudos por modelo de controle de acesso.

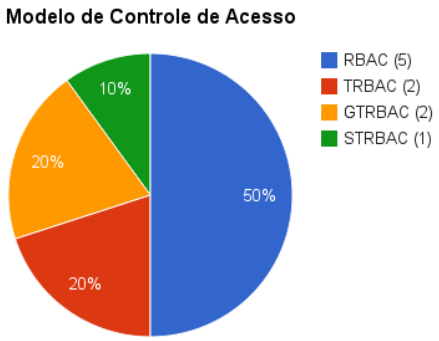


Figura 6.6: Modelos de Controle de Acesso

RQ2.1: Que tipos de problemas são tratados pela abordagem de teste?

Como pode ser visto na Tabela 6.2, os artigos selecionados majoritariamente tratam problemas relacionados a análise de segurança. Detecção de inconsistências, conflitos entre restrições e a análise de propriedades de segurança como vivacidade e segurança (safety) foram abordados. Além destes, três artigos tratam do problema de teste de políticas, mais especificamente teste de conformidade entre políticas e mecanismos de segurança (Masood et al., 2009, 2010a,b). Estes artigos foram publicados em periódicos de alto impacto (IEEE Transactions) e possuem um conjunto comum de três autores (Ammar Masood – Air University-Islamabad, Arif Ghafoor e Aditya P. Mathur – Purdue University-West Lafayette). Além disso, os dados coletados mostram estes artigos como os únicos discutindo teste de conformidade usando MEFs de sistemas de controle de acesso RBAC. Estas informações, junto do fato de que teste baseado em MEFs é uma área de pesquisa muito bem consolidada e com muitas questões de cunho teórico em aberto, evidenciam o Teste Baseado em MEFs de Controle de Acesso RBAC como um terreno

de muitas oportunidades de pesquisa. Uma possibilidade de pesquisa seria a replicação dos trabalhos considerando:

- **Critérios de Seleção:** Os trabalhos citados se limitaram ao uso de técnicas baseadas em busca, em requisitos (heurísticas) e especificação. Replicações poderiam ser conduzidas usando outros critérios, tais como baseado em fluxo de dados e similaridade (Bertolino et al., 2015). O mesmo vale para replicações usando diferentes ACUTs. Estudos com esse perfil permitiriam melhor entender o potencial e limitações do TBM usando MEFs de sistemas RBAC.
- **Tecnologia de Geração de Testes:** Essencialmente, os trabalhos se restringiram a um único algorítmo de geração baseado em busca e em geração aleatória. Uma série de algorítmos, como os discutidos no Capítulo 4, poderiam ser testados e comparados a fim de contribuir com ambos os campos de pesquisa, tanto de teste de políticas como de teste baseado em MEFs.

RQ2.2: Que políticas são usadas como exemplo?

Apenas nove dos dez artigos selecionados forneceram detalhes sobre as políticas utilizadas como exemplo. As políticas identificadas possuem diferentes quantidade de usuários, papéis e permissões e estão relacionadas a vários domínios de aplicação, tais como Hospitalar (Masood et al., 2009), Programa de Pontos de Clientes (Song e Chen, 2012), e Bancário (Geepalla et al., 2012). Bugliesi et al. (2012) omitiu informações sobre a política em teste por questões de sigilo.

RQ2.3: Que programas são usados como SUT?

Somente um dos artigos forneceu informações que deixassem evidente a aplicação dos casos de testes em mecanismos de política (Masood et al., 2009). O ACUT usado, X-GTRBAC, já foi discutido no Capítulo 5. Os demais artigos aparentemente se limitaram ao teste usando políticas, sem considerar softwares.

ID	Título	Autores	Ano	Origem	Tipo
A1	Formal Verification of Business Workflows and Role Based Access Control Systems	Dury, A. and Boroday, S. and Petrenko, A. and Lotz, V.	2007	International Conference on Emerging Security Information, Systems, and Technologies (SecureWare)	Conferência
A2	Roles-based Access Control Modeling and Testing for Web Applications	Bo Song and Shengbo Chen	2012	Third World Congress on Software Engineering (WCSE)	Conferência
A3	Scalable and Effective Test Generation for Role-Based Access Control Systems	Masood, A. and Bhatti, R. and Ghafoor, A. and Mathur, A.P.	2009	IEEE Transactions on Software Engineering	Periódico
A4	Fault coverage of Constrained Random Test Selection for access control: A formal analysis	Masood, A.a and Ghafoor, A.b and Mathur, A.P.c	2010	Journal of Systems and Software	Periódico
A5	Gran: Model Checking Grsecurity RBAC Policies	Bugliesi, M. and Calzavara, S. and Focardi, R. and Squarcina, M.	2012	IEEE 25th Computer Security Foundations Symposium (CSF)	Simpósio
A6	Security Analysis of Temporal-RBAC Using Timed Automata	Mondal, S. and Sural, S.	2008	Fourth International Conference on Information Assurance and Security (ISIAS)	Conferência
A7	Towards formal security analysis of GTRBAC using timed automata	Mondal, S.a and Sural, S.a and Atluri, V.b	2009	Proceedings of ACM Symposium on Access Control Models and Technologies (SACMAT)	Simpósio
A8	Security analysis of GTRBAC and its variants using model checking	Mondal, S.a and Sural, S.a and Atluri, V.b	2011	Computers & Security	Periódico
A9	Verification of Spatio-Temporal Role Based Access Control using Timed Automata	Geepalla, E. and Bordbar, B. and Okano, K.	2012	IEEE 3rd International Conference on Networked Embedded Systems for Every Application (NESEA)	Conferência
A10	Conformance Testing of Temporal Role-Based Access Control Systems	Masood, A. and Ghafoor, A. and Mathur, A.P.	2010	IEEE Transactions on Dependable and Secure Computing	Periódico

Tabela 6.1: Teste de Controle de Acesso Baseado em Estados - Artigos Selecionados

ID	Notação	Escopo	Caract.	Critério de Seleção	Tecnologia	Execução	Modelo de CA	Problema	Política	ACUT
A1	EFSM	E/S	UDD	Estrutural	Model Checking	Offline	RBAC	Análise de Segurança	Procure to Stock	N/A
A2	EFSM	E/S	UDD	Estrutural	Baseado em Busca	Offline	RBAC	Análise de Segurança	Experience points and Roles	N/A
A3	FSM	E/S	UDD	Estrutural, Requisitos, Especificação	Baseado em Busca, Aleatório	Offline	RBAC	Teste de Conformidade	Hospital	X-GTRBAC
A4	FSM	E/S	UDD	Estrutural, Requisitos, Especificação	Baseado em Busca, Aleatório	Offline	RBAC	Teste de Conformidade	2-Role, 1-User, 2-Permission	N/A
A5	LTS	E/S	UDD	Estrutural	Model Checking	Offline	RBAC	Análise de Segurança	N/A	N/A
A6	TA	E/S	TDD	Estrutural	Model Checking	Offline	TRBAC	Análise de Segurança	Full & Part Time Employee	N/A
A7	TA	E/S	TDD	Estrutural	Model Checking	Offline	GTRBAC	Análise de Segurança	4-Role, 16-User, 4-Permission	N/A
A8	TA	E/S	TDD	Estrutural	Model Checking	Offline	GTRBAC	Análise de Segurança	3-Role, 12-User, 5-Permission	N/A
A9	TA	E/S	TDD	Estrutural	Model Checking	Offline	STRBAC	Análise de Segurança	SECURE Banking	N/A
A10	TIOA	E/S	TDD	Estrutural	Baseado em Busca	Offline	TRBAC	Teste de Conformidade	Senior & Trainee Doctor	N/A

FSM: Finite State Machine | EFSM: Extended FSM | LTS: Labeled Transition System | TA: Timed Automata | TIOA: Timed Input-Output Automata | E/S: Entrada e Saída | CA: Controle de Acesso | UDD: Não Temporizado (Untimed), Determinístico, Discreto | TDD: Temporizado, Determinístico, Discreto

Tabela 6.2: Teste de Controle de Acesso Baseado em Estados - Dados Extraídos

6.6 Geração de Testes usando MEFs para Sistemas RBAC

Como discutido na Seção 6.2, sistemas de controle de acesso podem ser especificados na forma de sistemas de transição de estados. Além disso, na Seção 6.5 vimos que os trabalhos de Masood et al. (2009, 2010a,b) foram os únicos aplicando teste de conformidade usando máquinas de estados finitos em sistemas RBAC, sendo o primeiro (Masood et al., 2009) embasamento teórico para os demais. Por esta razão, as principais contribuições de Masood et al. (2009) serão apresentadas e discutidas a seguir.

Em Masood et al. (2009), uma política RBAC é apresentada na forma de uma 16-tupla $\mathcal{P} = (U, R, Pr, UR, PR, \leq_A, \leq_I, I, S_u, D_u, S_r, D_r, SSoD, DSoD, S_s, D_s)$, onde:

- U e R são, respectivamente, os conjuntos finitos de usuários, papeis;
- Pr é o conjuntos finito de permissões;
- $UR \subseteq U \times R$ é o conjunto de atribuições usuário-permissão admissíveis;
- $PR \subseteq Pr \times R$ é o conjunto de atribuições permissão-papel admissíveis;
- $\leq_A \subseteq R \times R$ e $\leq_I \subseteq R \times R$ são, respectivamente, o conjunto de relações de ativação e de hierarquia de papeis;
- $I = \{AS, DS, AC, DC, AP, DP\}$ é o conjunto de tipos de requisições possíveis de serem enviadas para um ACUT. Estas requisições possibilitam a atribuição (AS), desatribuição (DS), ativação (AC) e desativações (DC) de relações usuário-papel e a atribuição (AP) e desatribuição (DP) de relações permissão-papel;
- $S_u, D_u : U \rightarrow Z^+$ são restrições de cardinalidade estática (S_u) e dinâmica (D_u) em U onde Z^+ denota o conjunto de valores inteiros não negativos;
- $S_r, D_r : R \rightarrow Z^+$ são restrições de cardinalidade estática (S_r) e dinâmica (D_r) em R onde Z^+ denota o conjunto de valores inteiros não negativos;
- $SSoD, DSoD \subseteq 2^R$ são os conjuntos de segregação de funções (SoD) estática (SSoD) e dinâmica (DSoD);
- $S_s : SSoD \rightarrow Z^+$ especifica a cardinalidade dos conjuntos SSoD; e
- $D_s : DSoD \rightarrow Z^+$ especifica a cardinalidade dos conjuntos DSoD.

Um exemplo de política adaptado de (Masood et al., 2009) será descrito a seguir.

$$U = \{John, Mary\} = \{u_1, u_2\} \tag{6.1}$$
$$R = \{Cliente\} = \{r_1\} \tag{6.2}$$
$$Pr = \{Deposito, Saque\} = \{p_1, p_2\} \tag{6.3}$$
$$UR = \{(u_1, r_1), (u_2, r_1)\} \tag{6.4}$$
$$PR = \{(p_1, r_1), (p_2, r_1)\} \tag{6.5}$$
$$S_u(u_1) = S_u(u_2) = D_u(u_1) = D_u(u_2) = 1 \tag{6.6}$$
$$S_r(r_1) = 2 \tag{6.7}$$
$$D_r(r_1) = 1 \tag{6.8}$$
$$\leq_A = \leq_I = \{\} \tag{6.9}$$

A política P descrita acima possui dois usuários (6.1), um papel (6.2) e duas permissões (6.3). Ambos os usuários u_1 e u_2 estão atribuídos ao papel r_1 (6.4) que por sua vez está atribuído a ambas as permissões p_1 e p_2 (6.5). As restrições de cardinalidade estáticas e dinâmicas dos usuários foram definidas em 1 (6.6). A restrição de cardinalidade estática do papel r_1 foi estabelecida em 2 (6.7) e a dinâmica em 1 (6.8). Não foram estabelecidas relações de hierarquia ou ativação de hierarquia (6.9).

6.6.1 Modelagem de Políticas RBAC usando MEFs

A partir desta definição de política RBAC, Masood et al. (2009) define uma estratégia de modelagem de sistemas de controle de acesso usando máquinas de estado. Nesta estratégia o comportamento de um mecanismo de controle de acesso, que é estabelecido por uma política $\mathcal{P}$, é descrito como uma máquina de estados finitos MEF($\mathcal{P}$). Os estados de MEF($\mathcal{P}$), descrito como S, são nomeados usando uma sequência de pares de bits, havendo um par para cada combinação usuário-papel (Figura 6.3). O par de bits 01 não é utilizado pois não podem haver relações ativas e não atribuídas.

Tabela 6.3: Padrão de representação para relação usuário-papel em pares de bits

	Papel	
Relação / **Padrão**	**Atribuída**	**Ativa**
00	Não	Não
10	Sim	Não
11	Sim	Sim
01	-	-

O domínio de entrada X da MEF($\mathcal{P}$) assume a forma $X = Rq(up, r), Rq \in I, up \in (U \cup Pr), r \in R$. O domínio de saída Z se apresenta como $Z = \{granted, denied\}$. A função de

saída e a função de transição de estados, respectivamente denominadas $f_z : (S \times X) \rightarrow Z$ e $f_s : (S \times X) \rightarrow S$, são definida com base nas restrições da política $\mathcal{P}$, estado atual da MEF($\mathcal{P}$) e nas requisições recebidas.

Esta abordagem, em geral, apresenta como limite superior para o total de estados o valor de $3^{u \times r}$, tal que u é o total de usuários e r é o total de papéis. A MEF($\mathcal{P}$) referente a política $\mathcal{P}$ citada anteriormente pode ser vista na Figura 6.7. Para facilitar a visualização do modelo, foram omitidas as transições em *loop*.

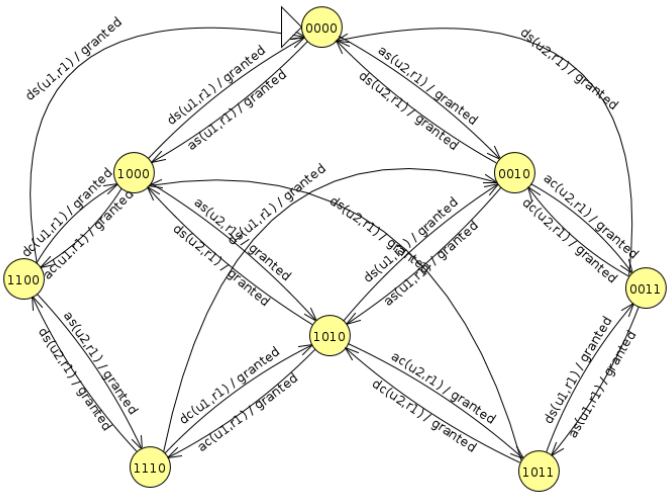


Figura 6.7: Máquina de Estados Completa derivada de uma Política RBAC

A MEF($\mathcal{P}$) da Figura 6.7 apresenta um total de 8 estados definindo as possíveis configurações que o ACUT pode assumir. O total de estados, inferior ao limite superior ($3^{2 \times 1} = 9$), irá variar de acordo com as restrições definidas na política $\mathcal{P}$. Por exemplo, o estado 1111 não é encontrado na MEF pois a restrição de cardinalidade dinâmica estabelecida para o papel r_1 (6.8) impede que hajam dois usuários com esse papel ativo simultaneamente.

6.6.2 Teste de Conformidade de RBAC usando MEFs

A partir dos conceitos citados, podemos definir um ambiente de testes onde uma MEF($\mathcal{P}$) descreve o comportamento de uma política $\mathcal{P}$ e é usada para testar, verificar e validar a conformidade entre mecanismo e política de controle de acesso. Na Figura 6.8 pode ser vista uma descrição visual do cenário de teste de conformidade de mecanismos de controle de acesso.

Considerando o conceito de desenvolvimento multi-fase, no teste de conformidade de mecanismos de controle de acesso uma MEF($\mathcal{P}$), gerada a partir de uma política $\mathcal{P}$, pode ser usada como especificação do comportamento ($ACUT_{\mathcal{P}}$) de um mecanismo de controle de acesso ($ACUT_M$). Esta política $\mathcal{P}$ é processada pelo $ACUT_M$ e uma representação interna de $\mathcal{P}$ é usada para implementar o controle de acesso que a política $\mathcal{P}$ estabelece. Dessa forma, com ajuda de um módulo de teste que automatize a geração de entradas, capture saídas e, se neces-

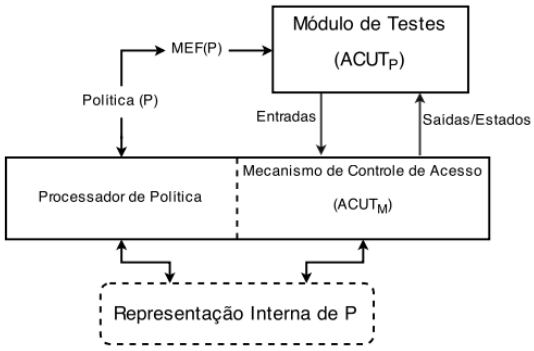


Figura 6.8: Teste de ACUTs

sário, consulte os estados de ambos os ACUTs a conformidade entre os $ACUT_M$ e $ACUT_{\mathcal{P}}$ pode ser verificada e validada.

Usando uma notação um mais formal, o objetivo do teste de conformidade de um ACUT pode ser definido como validar com o máximo de certeza as seguintes condições:

$$\forall Rq(up,r) \in I,\ S_{ACUT_{\mathcal{P}}}[Rq(up,r)] = S'_{ACUT_{\mathcal{P}}} \rightarrow S_{ACUT_M}[Rq(up,r)] = S'_{ACUT_M} = S'_{ACUT_{\mathcal{P}}} \tag{6.10}$$

$$\forall Rq(up,r) \in I_{\text{inválido}},\ S_{ACUT_M}[Rq(up,r)] = S_{ACUT_M} \tag{6.11}$$

A condição 6.10 diz respeito à equivalência comportamental entre $ACUT_M$ e $ACUT_{\mathcal{P}}$ considerando quaisquer requisições, sejam (des)atribuições ou (des)ativações de relações. A condição 6.11, por sua vez, diz respeito ao requisito de que um $ACUT_M$ em hipótese alguma deve apresentar um comportamento inesperado, ignorando requisições inválidas. Entende-se por requisição inválida todo aquele dado de entrada não descrito em uma especificação. Ao serem garantidas estas condições, podemos afirmar com um certo grau de certeza que um mecanismo de controle de acesso implementa uma política de segurança corretamente.

6.6.3 Critérios de Seleção para Teste de RBAC

Usando a estratégia de modelagem de ACUTs usando MEFs, discutida anteriormente, Masood et al. (2009) propõe e avalia o seguinte conjunto de técnicas para teste:

i) Um modelo de defeitos para RBAC baseado em mutação seletiva e defeitos maliciosos;

ii) Uma estratégia de teste de conformidade baseada em MEF completa;

iii) Um conjunto de seis heurísticas para teste de conformidade;

iv) Um procedimento de geração de teste denominado *Constrained Random Test Selection*; e

v) Uma técnica para teste funcional de ACUTs.

Enquanto a técnica de teste funcional (v) é descrita como critério de seleção baseado em cobertura de código (Masood et al., 2009), os quatro demais critérios podem ser caracterizados, considerando a taxonomia de Utting et al. (2012), da seguinte forma: Critério Baseado em Defeitos (i), Critério Baseado em Cobertura Estrutural (ii), Critério Baseado em Requisitos (iii) e Critério Baseado em Especificação de Caso de Teste (iv). A seguir, cada um dos critérios será melhor discutido.

6.6.3.1 Modelo de Defeitos para RBAC

Como o próprio nome sugere, o modelo de defeitos para RBAC proposto por Masood et al. (2009) é um critérios de seleção baseados em defeitos. Ele define duas técnicas de injeção de defeitos, uma usando Mutação Seletiva de Políticas RBAC e outra Baseada em Defeitos Maliciosos, descritas a seguir.

Mutação Seletiva

A mutação seletiva de políticas RBAC define que dada uma política RBAC, $\mathcal{P} \in \mathcal{R}$, o conjunto $\mathcal{R}$ pode ser resumido a $\mathcal{R} = \mathcal{R}^{\mathcal{P}}_{conforme} \cup \mathcal{R}^{\mathcal{P}}_{defeito}$. O subconjunto $\mathcal{R}^{\mathcal{P}}_{conforme}$ é constituido pelas políticas com comportamento equivalente a $\mathcal{P}$ enquanto que $\mathcal{R}^{\mathcal{P}}_{defeito}$ é formado pelas políticas defeituosas e que, em consequência, podem ser diferenciadas de $\mathcal{P}$ por um caso de teste.

A partir destes conceitos, podemos restringir o espaço de busca de casos de teste aqueles que conseguirem evidenciar que a política $\mathcal{P}$ não é defeituosa, ou seja, $\mathcal{P} \notin \mathcal{R}^{\mathcal{P}}_{defeito}$. Para isso foi proposto um conjunto de operadores que possibilita a geração de políticas mutantes $\mathcal{P}' = (U, R, Pr, UR', PR', \leq'_A, \leq'_I, I, S'_u, D'_u, S'_r, D'_r, SSoD', DSoD', S'_s, D'_s)$ dada uma política $\mathcal{P} = (U, R, Pr, UR, PR, \leq_A, \leq_I, I, S_u, D_u, S_r, D_r, SSoD, DSoD, S_s, D_s)$.

Como pode ser visto, o conjunto de usuários (U), papeis (R), permissões (Pr) e tipos de requisição (I) em $\mathcal{P}'$ é o mesmo de $\mathcal{P}$, enquanto que os elementos UR', PR', $\leq'_A$, $\leq'_I$, S'_u, D'_u, S'_r, D'_r, $SSoD'$, $DSoD'$, S'_s e D'_s são modificados. Os operadores propostos foram ser classificados em duas categorias: Operadores de Mmutação e Operadores de Modificação de Elementos.

Operadores de Mutação são aplicados sobre UR', PR', $\leq'_A$, $\leq'_I$, $SSoD'$ e $DSoD'$ a fim de realizar a adição, remoção ou modificação de elementos. A aplicação deste operador de mutação permite, por exemplo, a substituição de um usuário u_a com o papel r por outro u_b, transformando (u_a, r) em (u_b, r).

Operadores de Modificação de Elementos são aplicados sobre S'_u, D'_u, S'_r, D'_r, S'_s e D'_s permitindo o incremento ou decremento dos seus valores. A mutação usando esse operador permite, por exemplo, gerar uma política $\mathcal{P}'$ com cardinalidade de papel incrementada ou decrementada em 1.

Na Tabela 6.4 os operadores de mutação são mostrados associados aos tipos de defeitos que eles podem gerar num ACUT. Os tipos de defeitos gerados pelos operadores são organizados em

três tipos: Defeito de Atribuição (UR1 e UR2), Defeito de Ativação (UA1 e UA2) e Defeito de Permissão (PR1 e PR2).

Tabela 6.4: Impacto dos Operadores de Mutação no RBAC (Masood et al., 2009)

Estruturas Mutadas	**Possível Impacto no ACUT' (Defeito)**
UR', S'_u, S'_r, S'_s	UR1, UR2
PR', $\leq'_I$	PR1, PR2
$\leq'_A$, D'_u, D'_r, $DSoD'$, D'_s	UA1, UA2

Defeitos do tipo UR1 restringem usuários de serem atribuídos a um papel ou implicam em uma desatribiução não autorizada. Defeitos do tipo UR2 podem implicar em atribuições não autorizadas. Defeitos do tipo PR1 restringem uma permissão de ser associada a um papel ou causam uma desatribuição não autorizada. Defeitos do tipo PR2 atribuem permissões para papéis não autorizados. Defeitos do tipo UA1 e UA2 são semelhantes ao UR1 e UR2, porém, impactando na ativação de papéis. Cada tipo de defeito em RBAC também é apresentado associado ao modelo de defeitos de MEFs proposto por Chow (1978) (Tabela 6.5).

Tabela 6.5: Correspondência Entre o Modelo de Defeitos RBAC e de MEFs

Defeitos em RBAC (Masood et al., 2009)	**Defeitos em MEF (Chow, 1978)**
UR1, UA1, PR1	Erro de Transferência, Estados Ausente, Erro de Saída
UR2, UA2, PR2	Estados Extras, Erro de Saída, Erro de Transferência

Defeitos Maliciosos

Defeitos que não podem ser modelados como mutações de políticas RBAC são colocados na categoria de defeitos maliciosos. Estes defeitos são inseridos na tentativa de simular um programador mal intencionado implementando meios de burlar um mecanismo de controle de acesso. Três tipos de defeitos são citados:

- **Defeitos Baseados em Contador:** Um trecho de código pode ser implementado a fim de permitir que uma requisição seja garantida, negada ou abortada com base na contagem de eventos.
- **Defeitos Baseados em Entrada/Saída:** Um trecho de código pode ser implementado a fim de permitir que uma requisição inválida seja garantida, negada ou abortada com base em entradas específicas.
- **Defeitos Baseados em Sequência:** Um ou mais defeitos podem ser implementados no ACUT de modo que um acesso indevido seja permitido quando uma sequência específica de requisições for enviada.

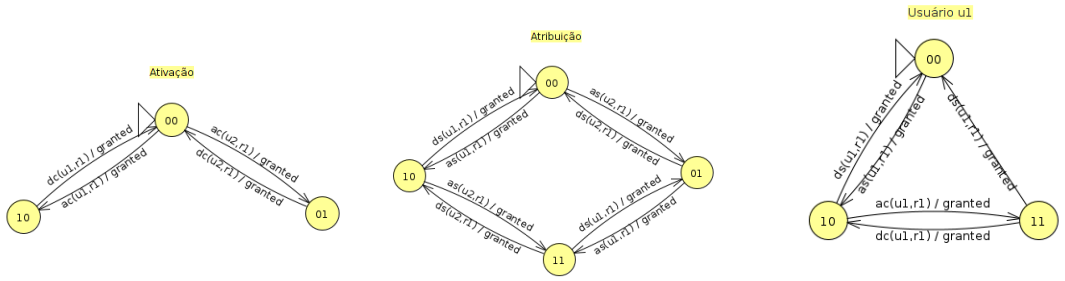
Enquanto defeitos baseados em contador podem ser relacionados a erros de estado extra, Defeitos Baseados em Entrada/Saída e Baseados em Sequência não podem ser simulados por meio de mutação de MEFs. Apesar disso, Defeitos Baseados em Sequência podem ser detectados por meio de quaisquer sequência que gere caminhos pelas MEFs (Masood et al., 2009).

6.6.3.2 Baseado em MEF Completa

Neste procedimento sequências de teste são geradas a partir de uma MEF completa MEF($\mathcal{P}$) que especifique uma política $\mathcal{P}$. Para isso, o conjunto *transition cover* é usado para gerar caminhos da raiz da árvore de testes até os seus ramos. Caso não seja possível observar os estados de uma MEF o método W (Chow, 1978) pode ser usado e, considerando uma estimativa correta do total de estados extras, todos os defeitos da MEF podem ser detectados. Com excessão dos Defeitos Baseados em Entrada/Saída, que podem se utilizar de entradas completamente desconhecidas, Defeitos Baseados em Contador e em Sequência também podem ser detectados. Entretanto, apesar da capacidade de detecção de falhas, esta abordagem sofre com o problema de explosão de estados, dado o limite superior máximo de $3^{u \times r}$. Masood et al. (2009) não recomenda o uso desta abordagem caso a combinação entre usuários e papéis exceda 20.

6.6.3.3 Baseado em Heurística

Um conjunto de seis heurísticas foi definido a fim de fornecer uma alternativa de teste que contorne o problema de explosão de estados. Por meio dessas heurísticas, ao invés de MEFs completas, são geradas MEFs descrevendo os estados das relações de ativação e atribuição separadamente. Essas heurísticas foram classificadas como critério de seleção baseado em requisitos (Utting et al., 2012) devido a sua relação com os requisitos funcionais do RBAC (usuários, papéis e tipos de requisição). As heurísticas e suas respectivas MEFs encontram-se ilustradas na Figura 6.9 e serão descritas a seguir.



(a) M_{ac} descrevendo ativações **(b)** M_{as} descrevendo atribuições **(c)** M_u descrevendo ativações e atribuições para usuário

Figura 6.9: MEFs geradas usando Heurísticas

H1 **Separação de atribuição e ativação**: Construir as MEFs M_{AS} e M_{AC_1}, $M_{AC_2} \ldots M_{AC_k}$, $k > 0$. M_{AS} é uma MEF que modela todas as solicitações de atribuição e, para cada

estado da MEF M_{AS}, uma MEF M_{AC} especifica todas as ativações das atribuições. Nas Figuras 6.9(b) e 6.9(a) podem ser vistas, respectivamente, as MEFs M_{AS} e M_{AC} da política exemplo da Seção 6.6.

H2 **Uma MEF para ativação e uma única sequência de teste para atribuição**: Construir uma MEF M_{AC} para requisições de ativação com relação a um único estado $q_{max} \in M_{AS}$. O estado q_{max} define os papeis atribuídos que deverão ser ativados usando uma única sequência de teste que leve de um estado inicial em M_{AS} até a ativação de todos os papeis atribuídos em q_{max}. A máquina de estados M_{AC} para a política exemplo da Seção 6.6 pode ser vista na Figura 6.9(a). Neste caso, o estado $q = 11$ não existe em M_{AC} pois há uma restrição de cardinalidade para a ativação do papel $r1$ (6.8).

H3 **Uma única sequência de teste para atribuição e ativação**: Uma única sequência é usada para testar atribuições, ativações, desativações e desatribuições considerando todos os pares usuário-papel em qualquer ordem. Neste caso, o comportamento do mecanismo de controle de acesso é testado usando uma mistura dos quatro tipos de requisições aplicadas sobre a MEF($\mathcal{P}$).

H4 **MEF para cada usuário**: Construir uma MEF M_u para cada $u \in U$ descrevendo todas as ativações e atribuições. A partir disso, o procedimento de geração de testes é aplicado para cada modelo. Na Figura 6.9(c) pode ser vistas a MEF M_{u_1} referente a ativação e atribuição de relações considerando um usuário $u_1 \in U$.

H5 **MEF para cada papel**: Construir uma MEF M_r para cada $r \in R$ descrevendo todas as ativações e atribuições. A partir disso, o procedimento de geração de testes é aplicado para cada modelo. A MEF M_r é semelhante a gerada para usuários (Figura 6.9(c)), com a excessão de que as ativações e atribuições são feitas considerendo u papel $r \in R$ e não um usuário.

H6 **Agrupamento de usuários**: São criados grupos de usuários $UG = \{UG_1, UG_2, \ldots, UG_k\}$, $k > 0$, tal que $U_{i=1}^{k} UG_i = U$ e $UG_i \cap UG_j = \emptyset, 1 \leq i, j \leq k, i \neq k$. As requisições enviadas a um grupo são utilizadas por todos os usuários pertencentes a ele. Os grupos podem ser criados considerando atributos comuns de usuários.

6.6.3.4 Constrained Random Test Selection

O método de geração *Constrained Random Test Selection* (CRTS) especifica um valor $k > 0$ como total de requisições a serem enviadas para um ACUT. A partir disso, um numero k de requisições é gerado aleatoriamente e aplicado na MEF completa a fim de determinar suas saídas e estados correspondentes. Cada requisição $r_i, 1 \leq r \leq k$ é gerada selecionando aleatoriamente um usuário $u \in U$, um papel $r \in R$ e o tipo de requisição $i \in I = \{AS, DS, AC, DC, AP, DP\}$ com base na política $\mathcal{P}$.

Essa abordagem permite diminuir o número de sequência de testes sem reduzir o tamanho do modelo. Sugere-se como valor para k o comprimento do caminho mais longo da árvore de

testes da MEF completa. Isso garante sequências de teste capazes de atravessar a árvore da raiz até um nó folha. O limite superior para k deve ficar limitado em $2|U||R| + |Pr||R| + 1$.

6.6.3.5 Teste Funcional de ACUTs

Diferentemente do teste de conformidade, que fornece o seu parecer com base em uma política RBAC específica, o teste funcional requer que a corretude do mecanismo seja assegurada considerando, quando possível, todo o domínio $\mathcal{R}$ de políticas existentes. Esse fato inviabiliza abordagens exaustivas e força o teste a se restringir somente um número finito de políticas. Em consequência, ao término do teste é possível que hajam partes do código do ACUT não cobertas. Neste contexto, critérios de seleção baseados em mutação de políticas ou cobertura de código podem ajudar a aumentar a garantia de corretude do ACUT.

Masood et al. (2009) propõe uma abordagem de teste funcional que usa um conjunto de políticas geradas, $P_{set} = \{P_1, P_2 \ldots, P_k\},\ k > 0$, e um conjunto de testes gerado a partir das políticas, $T_{set} = \{T_1, T_2 \ldots, T_k\}$. Esta abordagem de teste funcional é composta pelos seguintes passos:

1. Gerar um conjunto inicial P_{set}.
2. Derivar o conjunto T_{set} a partir de P_{set}.
3. Testar o ACUT usando T_{set} e remover falhas detectadas.
4. Avaliar T_{set} usando critérios de cobertura de código.
5. Se os critérios de seleção forem satisfeitos, encerrar.
6. Incrementar P_{set}, atualizar T_{set} e voltar para o passo 3.

No artigo, a geração do conjunto de políticas P_{set} é feita manualmente e não são fornecidas regras para a execução desse passo. Entretanto, é ressaltado que P_{set} deve ser composto por políticas significativamente representativas para o ACUT de modo que a cobertura de código seja maximizada. Além disso, a técnica de mutação de políticas é sugerida como alternativa para a ampliação do conjunto inicial P_{set}. A geração do conjunto T_{set} pode ser feita considerando qualquer uma das abordagens propostas por Masood et al. (2009). O critério de cobertura usado no artigo é o MC/DC (Rajan et al., 2008).

6.7 Considerações Finais

Neste capítulo foram introduzidos e discutidos os principais aspectos relacionados a teste de controle de acesso baseado em modelos. Como foi visto, modelos baseados em estados são notações bastante populares no teste de segurança baseado em modelos. Essas notações também têm sido usadas por pesquisadores para especificar, testar e analisar políticas e sistemas de

controle de acesso. Também foi identificada através de um mapeamento sistemático a carência de estudos voltados para teste de controle de acesso baseado em estados. Em especial, o Teste de Conformidade de Sistemas RBAC usando Máquinas de Estados Finitos foi identificado como um tópico de pesquisa com questões em aberto que podem ser exploradas e gerar contribuições científicas tanto para a área de segurança da informação como de teste de software baseado em modelos. A análise comparativa envolvendo diferentes algorítmos, por exemplo, é um tópico interessante de ser pesquisado. Isso ocorre pois possibilita a replicação de estudos experimentais já existentes no contexto de MEFs, como o de Endo e Simao (2013), para fins de revalidação de resultados e também permite a análise em maior profundidade de técnicas de teste para RBAC usando MEF, como a proposta por Masood et al. (2009) que apesar de ser uma contribuição significativa peca por não ter considerado diferentes cenários (métodos de geração de sequências). A partir disso, a realização de um estudo envolvendo ambos os aspectos de Comparação de Métodos para Teste de MEF (Endo e Simao, 2013) e Teste de Políticas RBAC usando MEFs (Masood et al., 2009, 2010b) pode ser definida como bastante pertinente dada a sua contribuição mútua para duas áreas de pesquisa pouco exploradas em conjunto.

Capítulo

7

Plano de Trabalho

Diversas organizações ao redor do mundo adotam mecanismos de segurança, tais como sistemas de controle de acesso, com o intuito de proteger seus dados e recursos computacionais de atacantes interessados em danificá-los ou ter acesso privilegiado. Entretanto, como discutido no Capítulo 5, ao mesmo tempo em que esses mecanismos ajudam a mitigar ameaças, eles também podem ser fonte de outras preocupações.

Vulnerabilidades em sistemas de controle de acesso podem ser exploradas por usuários mal intencionados e oferecer outros riscos que, se não forem apropriadamente tratados, podem invalidar contramedidas de segurança. A fim de identificar esses problemas, muitas técnicas voltadas para a análise e teste de políticas têm sido propostas e estudadas.

Na Seção 6.4 foram vistos uma série de abordagens voltadas para o teste de políticas de controle de acesso. Martin et al. (2006) fazem um paralelo entre o teste de software tradicional e o teste de políticas de controle de acesso. Martin e Xie (2007) propõem métricas para análise de cobertura e teste de mutação de políticas XACML. Hansen e Oleshchuk (2005) utiliza *model checking* no teste de conformidade de entre políticas e mecanismos de controle de acesso. Mais recentemente, em Bertolino et al. (2015), são apresentadas duas técnicas de priorização de casos de teste baseadas em similaridade entre políticas XACML. Apesar disso, por meio da revisão de literatura apresentada na Seção 6.5 foi identificada uma carência de trabalhos em Teste de Sistemas RBAC usando Modelos Baseados em Estados.

Masood et al. (2009, 2010b) são alguns dos poucos autores atualmente discutindo Teste de Sistemas RBAC usando Máquinas de Estados Finitos. Em seus dois artigos (Masood et al., 2009, 2010b) um modelo de defeitos para RBAC, uma estratégia de TBM usando em MEF completa, seis heurísticas e uma técnica aleatória foram propostos e avaliados. Entretanto, apesar das

contribuições significativas, os resultados obtidos não podem ser generalizados se considerarmos a ausência de experimentos incluindo outros métodos de geração de sequências baseados em busca, tais como os apresentados na Seção 4.3.2.

Em Teste Baseado em Máquinas de Estados Finitos, ao serem propostos novos métodos de geração de sequências, é comum a realização de estudos comparativos entre métodos tradicionais de geração de sequências. Luo et al. (1995) compara o método HSI com os métodos W e Wp. Simão e Petrenko (2009b) avalia o método P com base no comprimento das sequências geradas e cobertura de defeitos. Simão et al. (2012) compara o método SPY com o HSI. Trabalhos voltados para a análise comparativa experimental entre métodos de geração de sequências, tais como Dorofeeva et al. (2010, 2005b); Endo e Simao (2013); Simão et al. (2009), podem fornecer *insights* sobre a capacidade dos algorimos e cenários mais indicados para cada método. Endo e Simao (2013), por exemplo, identificaram diferenças nas características dos conjuntos de teste gerados pelos métodos H e SPY, quando comparados com o HSI.

A partir disso, considerando que a proposição de novos métodos de geração de sequências é um motivador para estudos comparativos envolvendo métodos tradicionais, foi identificado que um estudo envolvendo a abordagem de Masood et al. (2009) para teste de RBAC usando MEF completa e diferentes métodos de geração, assim como realizado por Endo e Simao (2013), forneceria um maior entendimento em profundidade quanto ao potencial do Teste Baseado em MEFs para Sistemas RBAC. Dessa forma, foi definido como objetivo desse trabalho analisar métodos mais recentes (H, SPY e P) e tradicionais (W e HSI) de geração de sequências para teste de máquinas de estados finitos com o propósito compará-los com relação as características, custo e efetividade dos conjuntos de teste gerados para Teste de Conformidade entre Sistemas e Políticas de Controle de Acesso Baseado em Papel.

Na Seção 7.1 desse capítulo será discutida a metodologia adotada nesse projeto. Na Seção 7.2 será apresentada a lista de atividades realizadas até o dado momento. Na Seção 7.3 será apresentado o cronograma de atividades e por ultimo, na Seção 7.4, serão apresentados os resultados esperados para esse trabalho de mestrado.

7.1 Metodologia

Contexto e Formulação de Hipóteses

Com a finalidade de investigar o potencial e limitações do Teste de Sistemas RBAC usando Máquinas de Estados Finitos, foi definida a realização de um estudo experimental combinando o foco de Masood et al. (2009) e Endo e Simao (2013). A partir disso, considerando as características, custo e eficiência dos conjuntos de teste obtidos usando métodos tradicionais e mais recentes de geração de sequências, as seguintes hipóteses foram formuladas:

H_0 : Métodos mais recentes e tradicionais de geração de sequências não apresentam características, custo e eficiência diferentes, mesmo considerando MEFs geradas a partir de políticas RBAC.

H_1 : Resultados diferentes podem ser obtidos usando cada um dos métodos, mesmo considerando MEFs geradas a partir de políticas RBAC.

Seleção de Variáveis e Objetos de Estudo

Para esse experimento, foram estabelecidos como variáveis dependentes as características, custo e eficiência dos conjuntos de teste, e como variáveis independentes os métodos de geração de sequências. Como instrumentos a serem usados durante o experimento, foram definidos os seguintes elementos:

1. O conjunto de políticas RBAC coletadas pelo mapeamento sistemático (Seção 6.5);
2. Uma ferramenta de conversão de políticas RBAC para MEFs;
3. Uma ferramenta de mutação seletiva para políticas RBAC; e
4. Os métodos de geração de sequências de teste W, HSI, H, SPY e P.

As **políticas dos estudos identificados** no mapeamento da Seção 6.5 serão analisadas e incluídas no estudo. Para isso, o número de usuários (U), papéis (R) e permissões (Pr), uso de relações (UR, Pr, $\leq_A$ e $\leq_I$), de e restrições de cardinalidade (S_u, D_u, S_r e D_r) e de Segregação de Função ($SSoD$, $DSoD$, S_s e D_s) será avaliado. Se for verificado que algum tipo de restrição RBAC não está sendo utilizado ou que as políticas não sejam consideradas representativas, pesquisadores e profissionais de segurança da informação serão convidados para contribuir com esse estudo.

Uma ferramenta para conversão de políticas RBAC para Máquinas de Estados Finitos, denominada **rbac2fsm**, será codificada com base no artigo de Masood et al. (2009). Esta ferramenta, além de converter políticas RBAC em MEFs, também permitirá mapear restrições de políticas RBAC e transições da MEF. Esse mapeamento será usado a fim de possibilitar a rastreabilidade entre sequências de teste e elementos e restrições da política RBAC.

Uma ferramenta para mutação seletiva de políticas RBAC, denominada **rbacMutation**, também será codificada a fim de possibilitar a geração de políticas mutantes usando os operadores propostos por Masood et al. (2009).

Para os métodos de geração de sequências W, HSI, H, SPY e P serão utilizados o *lab package* de Endo e Simao (2013) e a ferramenta **JPLAVIS**. Caso seja necessário, os métodos também poderão ser reimplementados.

Considerando os elementos citados acima e o esquema do processo de Teste Baseado em Modelos (Figura 3.1) foi definido o método listado a seguir e ilustrado na Figura 7.1:

(1) Análise, Refinamento e Documentação do Conjunto de Políticas RBAC em Teste ($\mathcal{P}_{Set}$);

(2) Projeto e Implementação da ferramenta **rbac2fsm**;

(3) Projeto e Implementação da ferramenta **rbacMutation**;

(4) Geração das Políticas Mutantes $\mathcal{P}'$;

(5) Geração das Máquinas de Estados $MEF(P)$;

(6) Geração de Sequências e Teste das MEFs usando os Métodos W, HSI, H, SPY e P;

(7) Análise da Cobertura dos Modelos e dos Testes;

(8) Análise da Rastreabilidade entre Testes e Políticas.

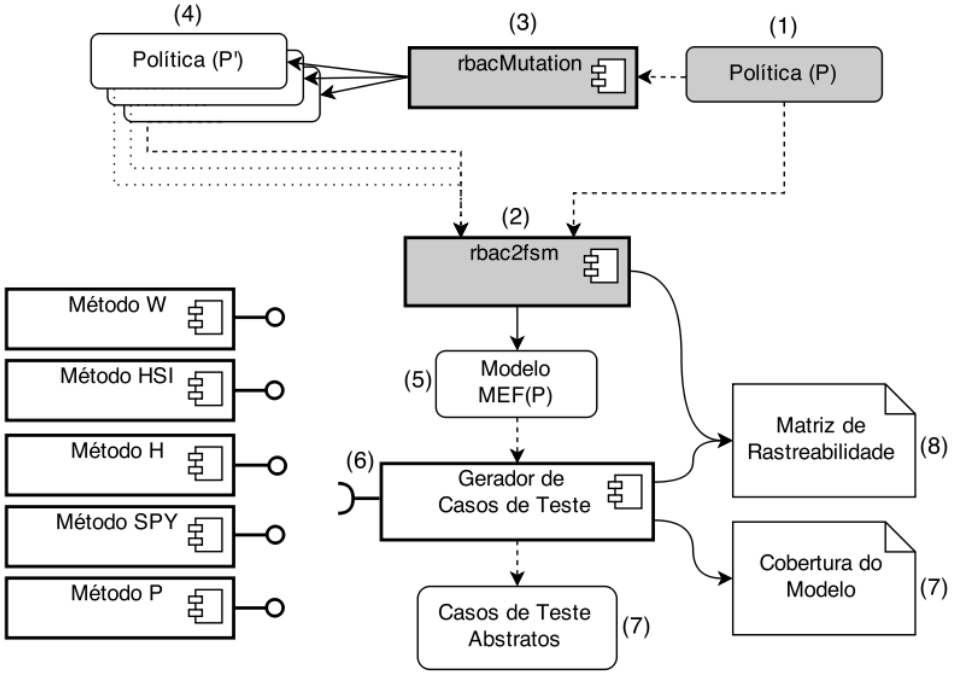


Figura 7.1: Esquema do Método

Primeiramente, o conjunto de Políticas RBAC (1) será avaliado e documentado, se julgado necessário ele passará por um refinamento. Após isso, as ferramentas (2) **rbac2fsm** e (3) **rbacMutation** serão projetadas e implementadas. Para auxiliar nesse processo, será considerado o uso de frameworks RBAC, tais como os apresentados no Capítulo 5. Após isso, (4) as políticas P serão repassadas para a ferramenta **rbacMutation** a fim de serem geradas políticas mutantes ($\mathcal{P}'$). As políticas originais e mutantes serão (5) convertidas para MEFs com auxílio da ferramenta **rbac2fsm**. Ao ser obtido o conjunto de MEF(P) (6) os métodos de geração serão executados e, em seguida, (7) os seus resultados analisados. Além disso, será feita uma (8) rastreabilidade entre os casos de teste e as políticas testadas. Mais especificamente, essa rastreabilidade será efetuada a fim de analisar o impacto dos casos de teste sobre as restrições da política RBAC e consequentemente também sobre o modelo de defeitos RBAC de Masood

et al. (2009). A análise dos casos de teste e da cobertura dos modelos será efetuada da mesma forma como em Endo e Simao (2013). O procedimento a ser usado na análise de rastreabilidade, entretanto, ainda está em aberto.

7.2 Atividades Realizadas

Durante o primeiro e segundo semestre o mestrando cursou as disciplinas necessárias para o cumprimento dos 51 créditos mínimos exigidos para depósito da dissertação. Além disso, ele foi aprovado com conceito A em todas elas. As seguintes disciplinas foram cursadas:

- Tópicos em Computação e Matemática Computacional I (1^o Semestre; 1 Crédito);
- Engenharia de Software (1^o Semestre; 12 Créditos);
- Preparação Pedagógica (1^o Semestre; 4 Créditos);
- Revisão Sistemática em Engenharia de Software (1^o Semestre; 6 Créditos);
- Engenharia de Software Experimental (1^o Semestre; 6 Créditos);
- Metodologia de Pesquisa Científica em Computação (2^o Semestre; 2 Créditos);
- Validação e Teste de Software (2^o Semestre; 12 Créditos); e
- Reúso de Software (2^o Semestre; 12 Créditos).

Durante a Disciplina de *Revisão Sistemática em Engenharia de Software* o aluno desenvolveu como projeto final uma revisão sistemática no contexto de Teste de Segurança Baseado em Modelos. Essa revisão foi submetida, aceita e apresentada pelo próprio aluno no *8th Brazilian Workshop on Systematic and Automated Software Testing* (SAST 2014) ocorrido em Maceió-AL. O artigo pode ser encontrado nos anais do evento[1]. Os resultados obtidos com essa revisão serviram de embasamento para essa qualificação e para a realização de um segundo estudo sistemático. Uma parte dos resultados de ambos os estudos pode ser verificada na Seção 6.5. O aluno também realizou e foi aprovado no exame de proficiência em inglês ainda durante o primeiro semestre.

Durante esse período de um ano, o aluno teve contato com artigos, livros e relatórios técnicos voltados para diversos assuntos, tais como Teste de Software, Teste Baseado em Modelos, Teste Baseado em Máquinas de Estados Finitos, Engenharia de Software Experimental, Controle de Acesso e RBAC.

[1] `http://www.ic.ufal.br/evento/cbsoft2014/anais/sast_v1_p.pdf`

7.3 Cronograma de Atividades

Para alcançar o objetivo definido, foi estabelecido um conjunto de atividades listado a seguir e ilustrado na Tabela 7.1.

1. Obtenção dos créditos exigidos para o mestrado
2. Redação da Qualificação de Mestrado
3. Exame de proficiência
4. Estudos sobre Teste Baseado em Modelos
5. Estudos sobre Controle de Acesso
6. Estudos sistemáticos sobre TBM e Controle de Acesso
7. Projeto do Experimento
8. Execução dos Experimentos
9. Redação da Dissertação
10. Defesa do mestrado

Tabela 7.1: Cronograma de projeto

	2014					2015						2016	
Atividades	Mar-Abr	Mai-Jun	Jul-Ago	Set-Out	Nov-Dez	Jan-Fev	Mar-Abr	Mai-Jun	Jul-Ago	Set-Out	Nov-Dez	Jan-Fev	Mar-Abr
1													
2													
3													
4													
5													
6													
7													
8													
9													
10													

7.4 Resultados Esperados

Por meio do experimento apresentado nesse capítulo, espera-se **(1) refutar a hipótese nula** H_0 da mesma forma como feito por Endo e Simao (2013). Os artefatos produzidos durante o experimento também fazem parte dos resultados esperados pois planeja-se disponibilizá-los na forma de **(2) lab package** a fim de permitir a replicação do estudo por terceiros.

Espera-se também a obtenção de dados experimentais que possibilitem a realização de investigações voltadas para **(3) análise de rastreabilidade em Teste de Sistemas RBAC usando MEFs**. A partir dessa análise de rastreabilidade, acredita-se que *insights* possam ser obtidos e permitam **(4) propor e investigar critérios de seleção e tecnologias de geração de testes específicos para o modelo RBAC**. Vale ressaltar que este ultimo resultado esperado não foi incluído como objetivo deste trabalho pois ainda não se tem certeza se ele é de fato tangível.

Referências Bibliográficas

Ali, S.; Briand, L.; Hemmati, H.; Panesar-Walawege, R. A systematic review of the application and empirical investigation of search-based test case generation. *IEEE Transactions on Software Engineering*, v. 36, n. 6, p. 742–762, 2010.

Alturi, V.; Ferraiolo, D. Role-based access control. In: van Tilborg, H. C. A.; Jajodia, S., eds. *Encyclopedia of Cryptography and Security*, Springer US, p. 1053–1055, 2011.

Ammann, P.; Offutt, J. *Introduction to software testing.* Cambridge University Press, 2008.

Anderson, R. *Security engineering: A guide to building dependable distributed systems.* Wiley, 2008.

Andrews, J.; Briand, L.; Labiche, Y.; Namin, A. Using mutation analysis for assessing and comparing testing coverage criteria. *IEEE Transactions on Software Engineering*, v. 32, n. 8, p. 608–624, 2006.

ANSI Role based access control. ANSI/INCITS 359-2004, 2004.

Apache Apache fortress. `http://directory.apache.org/fortress/`, acessado em 19 de Janeiro de 2015, 2014a.

Apache Apache redback. `http://archiva.apache.org/redback/`, acessado em 19 de Janeiro de 2015, 2014b.

Avritzer, A.; Larson, B. Load testing software using deterministic state testing. In: *Proceedings of the 1993 ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA '93, New York, NY, USA: ACM, 1993, p. 82–88 (*ISSTA '93*,).

Baier, C.; Katoen, J.-P. *Principles of model checking (representation and mind series).* The MIT Press, 2008.

Baumgrass, A.; Baier, T.; Mendling, J.; Strembeck, M. Conformance checking of rbac policies in process-aware information systems. In: Daniel, F.; Barkaoui, K.; Dustdar, S., eds. *Business Process Management Workshops*, v. 100 de *Lecture Notes in Business Information Processing*, Springer Berlin Heidelberg, p. 435–446, 2012.

Beizer, B. *Software testing techniques (2nd ed.)*. New York, NY, USA: Van Nostrand Reinhold Co., 1990.

Bertolino, A. Software testing research: Achievements, challenges, dreams. In: *Future of Software Engineering (FOSE 2007)*, 2007, p. 85–103.

Bertolino, A.; Daoudagh, S.; Kateb, D. E.; Henard, C.; Traon, Y. L.; Lonetti, F.; Marchetti, E.; Mouelhi, T.; Papadakis, M. Similarity testing for access control. *Information and Software Technology*, v. 58, n. 0, p. 355 – 372, 2015.

Bhatti, R.; Ghafoor, A.; Bertino, E.; Joshi, J. B. D. X-gtrbac: An xml-based policy specification framework and architecture for enterprise-wide access control. *ACM Transactions on Information and System Security*, v. 8, n. 2, p. 187–227, 2005.

Bozic, J.; Wotawa, F. Xss pattern for attack modeling in testing. In: *8th International Workshop on Automation of Software Test (AST 2013)*, 2013, p. 71–74.

Broy, M.; Jonsson, B.; Katoen, J.-P.; Leucker, M.; Pretschner, A. *Model-based testing of reactive systems: Advanced lectures (lecture notes in computer science)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.

Budd, T.; Angluin, D. Two notions of correctness and their relation to testing. *Acta Informatica*, v. 18, n. 1, p. 31–45, 1982.

Bugliesi, M.; Calzavara, S.; Focardi, R.; Squarcina, M. Gran: Model checking grsecurity rbac policies. In: *IEEE 25th Computer Security Foundations Symposium (CSF 2012)*, 2012, p. 126–138.

Chadwick, D. W.; Otenko, A. The {PERMIS} x.509 role based privilege management infrastructure. *Future Generation Computer Systems*, v. 19, n. 2, p. 277 – 289, selected Papers from the {TERENA} Networking Conference 2002, 2003.

Chow, T. S. Testing software design modeled by finite-state machines. *IEEE Transactions on Software Engineering*, v. 4, n. 3, p. 178–187, 1978.

Collins, L. Chapter 61 - access controls. In: Vacca, J. R., ed. *Computer and Information Security Handbook (Second Edition)*, second edition ed, Boston: Morgan Kaufmann, p. 1015 – 1021, 2013.

Colombo, M.; Lazouski, A.; Martinelli, F.; Mori, P. Access and usage control in grid systems. In: Stavroulakis, P.; Stamp, M., eds. *Handbook of Information and Communication Security*, Springer Berlin Heidelberg, p. 293–308, 2010.

Damasceno, C. D. N.; Delamaro, M. E.; Simão, A. d. S. Uma revisão sistemática em teste de segurança baseado em modelos. In: *Anais do Workshop Brasileiro de Testes de Software Automatizados e Sistemático - CBSoft - Congresso Brasileiro de Software: Teoria e Prática*, Porto Alegre: SBC, 2014, p. 31–40.

Delamaro, M. E.; Maldonado, J. C.; Jino, M. *Introdução ao teste de software*. Campus. Elsevier, 2007.

Demillo, R.; Lipton, R.; Sayward, F. Hints on test data selection: Help for the practicing programmer. *Computer*, v. 11, n. 4, p. 34–41, 1978.

Deutsch, M. Tutorial series 7 software project verification and validation. *Computer*, v. 14, n. 4, p. 54–70, 1981.

Do, H.; Rothermel, G. On the use of mutation faults in empirical assessments of test case prioritization techniques. *IEEE Trans. Softw. Eng.*, v. 32, n. 9, p. 733–752, 2006.

Dorofeeva, R.; El-Fakih, K.; Maag, S.; Cavalli, A. R.; Yevtushenko, N. Fsm-based conformance testing methods: A survey annotated with experimental evaluation. *Information and Software Technology*, v. 52, n. 12, p. 1286 – 1297, 2010.

Dorofeeva, R.; El-Fakih, K.; Yevtushenko, N. An improved conformance testing method. In: *Formal Techniques for Networked and Distributed Systems*, Springer, 2005a, p. 204–218 (*Lecture Notes in Computer Science*, v.3731).

Dorofeeva, R.; Yevtushenko, N.; El-Fakih, K.; Cavalli, A. R. Experimental evaluation of fsm-based testing methods. In: *Third IEEE International Conference on Software Engineering and Formal Methods (SEFM 2005)*, IEEE Computer Society, 2005b, p. 23–32.

Dury, A.; Boroday, S.; Petrenko, A.; Lotz, V. Formal verification of business workflows and role based access control systems. In: *The International Conference on Emerging Security Information, Systems, and Technologies (SecureWare 2007)*, 2007, p. 201–210.

Elliott, A.; Knight, S. Role explosion: Acknowledging the problem. In: Arabnia, H. R.; Reza, H.; Deligiannidis, L.; Cuadrado-Gallego, J. J.; Schmidt, V.; Solo, A. M. G., eds. *Software Engineering Research and Practice*, CSREA Press, 2010, p. 349–355.

Endo, A. T.; Simao, A. Evaluating test suite characteristics, cost, and effectiveness of fsm-based testing methods. *Information and Software Technology*, v. 55, n. 6, p. 1045 – 1062, 2013.

Fabbri, S.; Maldonado, J.; Delamaro, M. Proteum/fsm: a tool to support finite state machine validation based on mutation testing. In: *XIX International Conference of the Chilean Computer Science Society (SCCC 1999)*, 1999, p. 96–104.

Gargantini, A. 4 conformance testing. In: Broy, M.; Jonsson, B.; Katoen, J.-P.; Leucker, M.; Pretschner, A., eds. *Model-Based Testing of Reactive Systems*, v. 3472 de *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, p. 87–111, 2005.

Geepalla, E.; Bordbar, B.; Okano, K. Verification of spatio-temporal role based access control using timed automata. In: *IEEE 3rd International Conference on Networked Embedded Systems for Every Application (NESEA 2012)*, 2012, p. 1–6.

Ghezzi, C.; Jazayeri, M.; Mandrioli, D. *Fundamentals of software engineering.* 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2002.

Gill, A. *Introduction to the theory of finite state machines.* New York: McGraw-Hill, 1962.

Guttman, B.; Roback, E. A. *Sp 800-12. an introduction to computer security: The nist handbook.* Relatório Técnico, Gaithersburg, MD, United States, 1995.

Hansen, F.; Oleshchuk, V. Conformance checking of rbac policy and its implementation. In: *Proceedings of the First International Conference on Information Security Practice and Experience*, ISPEC'05, Berlin, Heidelberg: Springer-Verlag, 2005, p. 144–155 (*ISPEC'05*,).

Hu, V.; Martin, E.; Hwang, J.; Xie, T. Conformance checking of access control policies specified in xacml. In: *31st Annual International Computer Software and Applications Conference (COMPSAC 2007)*, 2007, p. 275–280.

Husted, T.; Massol, V. *Junit in action.* Manning Publications, 2004.

Huth, M. Formal methods and access control. In: van Tilborg, H. C. A.; Jajodia, S., eds. *Encyclopedia of Cryptography and Security*, Springer, p. 494–495, 2011.

IEEE Ieee standard glossary of software engineering terminology. *IEEE Std 610.12-1990*, p. 1–84, 1990.

IEEE Ieee standard classification for software anomalies. *IEEE Std 1044-1993*, 1994.

Jaeger, T. Reference monitor. In: van Tilborg, H. C. A.; Jajodia, S., eds. *Encyclopedia of Cryptography and Security*, Springer US, p. 1038–1040, 2011.

Jaeger, T.; Tidswell, J. E. Rebuttal to the nist rbac model proposal. In: *Proceedings of the Fifth ACM Workshop on Role-based Access Control*, RBAC '00, New York, NY, USA: ACM, 2000, p. 65–66 (*RBAC '00*,).

Jang-Jaccard, J.; Nepal, S. A survey of emerging threats in cybersecurity. *Journal of Computer and System Sciences*, v. 80, n. 5, p. 973 – 993, special Issue on Dependable and Secure Computing The 9th IEEE International Conference on Dependable, Autonomic and Secure Computing, 2014.

Jia, Y.; Harman, M. An analysis and survey of the development of mutation testing. *Software Engineering, IEEE Transactions on*, v. 37, n. 5, p. 649–678, 2011.

Jonsson, B. 21 finite state machines. In: Broy, M.; Jonsson, B.; Katoen, J.-P.; Leucker, M.; Pretschner, A., eds. *Model-Based Testing of Reactive Systems*, v. 3472 de *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, p. 611–614, 2005.

Kitchenham, B.; Charters, S. *Guidelines for performing systematic literature reviews in software engineering.* Relatório Técnico, Keele University and Durham University Joint Report, 2007. Disponível em `http://www.dur.ac.uk/ebse/resources/Systematic-reviews-5-8.pdf`

Lebeau, F.; Legeard, B.; Peureux, F.; Vernotte, A. Model-based vulnerability testing for web applications. In: *IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops (ICSTW 2013)*, 2013, p. 445–452.

Lee, D.; Yannakakis, M. Principles and methods of testing finite state machines-a survey. *Proceedings of the IEEE*, v. 84, n. 8, p. 1090–1123, 1996.

Li, N. Discretionary access control. In: van Tilborg, H.; Jajodia, S., eds. *Encyclopedia of Cryptography and Security*, Springer US, p. 353–356, 2011.

Li, N.; Tripunitara, M. V. Security analysis in role-based access control. *ACM Transactions on Information and System Security*, v. 9, n. 4, p. 391–420, 2006.

Linkman, S.; Vincenzi, A.; Maldonado, J. An evaluation of systematic functional testing using mutation testing. In: *Proceedings of VII International Conference on Empirical Assessment in Software Engineering, Staffordshire–UK*, 2003, p. 1–15.

Luo, G.; Petrenko, A.; v. Bochmann, G. Selecting test sequences for partially-specified nondeterministic finite state machines. In: Mizuno, T.; Higashino, T.; Shiratori, N., eds. *Protocol Test Systems*, IFIP - The International Federation for Information Processing, Springer US, p. 95–110, 1995.

Martin, E.; Xie, T. A fault model and mutation testing of access control policies. In: *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, New York, NY, USA: ACM, 2007, p. 667–676 (*WWW '07*,).

Martin, E.; Xie, T.; Yu, T. Defining and measuring policy coverage in testing access control policies. In: Ning, P.; Qing, S.; Li, N., eds. *Information and Communications Security*, Springer Berlin Heidelberg, 2006, p. 139–158 (*Lecture Notes in Computer Science*, v.4307).

Masood, A.; Bhatti, R.; Ghafoor, A.; Mathur, A. P. Scalable and effective test generation for role-based access control systems. *IEEE Transactions on Software Engineering*, v. 35, n. 5, p. 654–668, 2009.

Masood, A.; Ghafoor, A.; Mathur, A. Conformance testing of temporal role-based access control systems. *IEEE Transactions on Dependable and Secure Computing*, v. 7, n. 2, p. 144–158, 2010a.

Masood, A.; Ghafoor, A.; Mathur, A. Fault coverage of constrained random test selection for access control: A formal analysis. *Journal of Systems and Software*, v. 83, n. 12, p. 2607 – 2617, {TAIC} {PART} 2009 - Testing: Academic & Industrial Conference - Practice And Research Techniques, 2010b.

McCabe, T. J. A complexity measure. *IEEE Transactions on Software Engineering*, v. 2, n. 4, p. 308–320, 1976.

Mondal, S.; Sural, S. Security analysis of temporal-rbac using timed automata. In: *Fourth International Conference on Information Assurance and Security (ISIAS 2008)*, 2008, p. 37–40.

Mondal, S.; Sural, S.; Atluri, V. Towards formal security analysis of gtrbac using timed automata. In: *Proceedings of the 14th ACM Symposium on Access Control Models and Technologies*, SACMAT '09, New York, NY, USA: ACM, 2009, p. 33–42 (*SACMAT '09*,).

Mondal, S.; Sural, S.; Atluri, V. Security analysis of gtrbac and its variants using model checking. *Computers & Security*, v. 30, n. 2-3, p. 128 – 147, special Issue on Access Control Methods and Technologies, 2011.

Myers, G. J.; Sandler, C. *The art of software testing.* John Wiley & Sons, 2004.

Nakamura, E.; de Geus, P. *Segurança de redes em ambientes cooperativos:.* Novatec, 2007.

Open Source Security grsecurity. `https://grsecurity.net/`, acessado em 19 de Janeiro de 2015, 2014.

Ouerdi, N.; Azizi, M.; louis Lanet, J.; Azizi, A.; Ziane, M. Emv card: Generation of test cases based on sysml models. *IERI Procedia*, v. 4, n. 0, p. 133 – 138, 2013 International Conference on Electronic Engineering and Computer Science (EECS 2013), 2013.

OWASP Open web application security project. `https://owasp.org/`, acessado em 19 de Janeiro de 2015, 2014a.

OWASP Owasp php-rbac project. `https://owasp.org/index.php/OWASP_PHPRBAC_Project/`, acessado em 19 de Janeiro de 2015, 2014b.

Pinheiro, A. C. Subsídios para a aplicação de métodos de geração de casos de testes baseados em máquinas de estados. Dissertação de Mestrado, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2012.

Power, D.; Slaymaker, M.; Simpson, A. Automatic conformance checking of role-based access control policies via alloy. In: Erlingsson, U.; Wieringa, R.; Zannone, N., eds. *Engineering Secure Software and Systems*, v. 6542 de *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, p. 15–28, 2011.

Pressman, R. S. *Software engineering: A practitioner's approach.* 5th ed. McGraw-Hill Higher Education, 2001.

Rajan, A.; Whalen, M.; Heimdahl, M. The effect of program and model structure on mc/dc test adequacy coverage. In: *ACM/IEEE 30th International Conference on Software Engineering (ICSE 2008)*, 2008, p. 161–170.

Rapps, S.; Weyuker, E. J. Data flow analysis techniques for test data selection. In: *Proceedings of the 6th International Conference on Software Engineering*, Los Alamitos, CA, USA: IEEE Computer Society Press, 1982, p. 272–278.

Rashid, A.; Ramdhany, R.; Edwards, M.; Kibirige, S. M.; Babar, A.; Hutchison, D.; Chitchyan, R. Detecting and preventing data exfiltration report. `http://goo.gl/epK048`, acessado em 10 de maio de 2014, 2013.

Samarati, P.; Vimercati, S. Access control: Policies, models, and mechanisms. In: Focardi, R.; Gorrieri, R., eds. *Foundations of Security Analysis and Design*, v. 2171 de *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, p. 137–196, 2001.

Shabtai, A.; Elovici, Y.; Rokach, L. *A survey of data leakage detection and prevention solutions.* Springer Briefs in Computer Science. Springer, i-viii, 1-92 p., 2012.

Shafique, M.; Labiche, Y. A systematic review of state-based test tools. *International Journal on Software Tools for Technology Transfer*, v. 17, n. 1, p. 59–76, 2013.

Simão, A.; Petrenko, A.; Maldonado, J. C. Comparing finite state machine test coverage criteria. *IET Software*, v. 3, p. 91–105(14), 2009.

Simão, A.; Petrenko, A.; Yevtushenko, N. On reducing test length for fsms with extra states. *Software Testing Verification and Reliability*, v. 22, n. 6, p. 435–454, 2012.

Simão, A. S.; Petrenko, A. Fault Coverage-Driven Incremental Test Generation. *The Computer Journal*, 2009a.

Simão, A. S.; Petrenko, A. Fault coverage-driven incremental test generation. In: *5th Workshop on Advances in Model Based Testing (A-MOST 2009)*, 2009b, p. 1–10.

Simão, A. D. S.; Ambrósio, A. M.; Fabbri, S. C. P. F.; Amaral, A. S. M. S.; Martins, E.; Maldonado, J. C. Plavis/fsm: an environment to integrate fsm-based testing tools. 2005.

Sommerville, I. *Software engineering.* 9th ed. USA: Addison-Wesley Publishing Company, 2010.

Song, B.; Chen, S. Roles-based access control modeling and testing for web applications. In: *Third World Congress on Software Engineering (WCSE 2012)*, 2012, p. 57–62.

Stallings, W.; Brown, L. *Computer security: Principles and practice.* 1st ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2007.

Traon, Y. L.; Mouelhi, T.; Baudry, B. Testing security policies: Going beyond functional testing. In: *The 18th IEEE International Symposium on Software Reliability (ISSRE 2007)*, 2007, p. 93–102.

Upadhyaya, S. Mandatory access control. In: van Tilborg, H.; Jajodia, S., eds. *Encyclopedia of Cryptography and Security*, Springer US, p. 756–758, 2011.

Utting, M.; Legeard, B. *Practical model-based testing: A tools approach.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007.

Utting, M.; Pretschner, A.; Legeard, B. A taxonomy of model-based testing approaches. *Software Testing, Verification and Reliability*, v. 22, n. 5, p. 297–312, 2012.

Vincenzi, A.; Delamaro, M.; Höhn, E.; Maldonado, J. C. Functional, control and data flow, and mutation testing: Theory and practice. In: Borba, P.; Cavalcanti, A.; Sampaio, A.; Woodcook, J., eds. *Testing Techniques in Software Engineering*, v. 6153 de *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, p. 18–58, 2010.

Wallace, D.; Fujii, R. Software verification and validation: an overview. *Software, IEEE*, v. 6, n. 3, p. 10–17, 1989.

Weyuker, E. J. On testing non-testable programs. *The Computer Journal*, v. 25, n. 4, p. 465–470, 1982.

Xu, D.; Tu, M.; Sanford, M.; Thomas, L.; Woodraska, D.; Xu, W. Automated security test generation with formal threat models. *IEEE Transactions on Dependable and Secure Computing*, v. 9, n. 4, p. 526–540, 2012.

Zurko, M. E.; Simon, R. T. Separation of duties. In: van Tilborg, H.; Jajodia, S., eds. *Encyclopedia of Cryptography and Security*, Springer US, p. 1182–1185, 2011.