Teste de Software Baseado em Máquinas de Estados Finitos – Uma Revisão

Marcelo Fantinato
marcelof@cpqd.com.br

CPqD Telecom & IT Solutions (http://www.cpqd.com.br)
DSB – Diretoria de Soluções em *Billing*
Rod. Campinas – Mogi-Mirim (SP-340), Km 118,5
13086-902 – Campinas – SP – Brasil

UNICAMP – Universidade Estadual de Campinas (http://www.unicamp.br)
FEEC – Faculdade de Engenharia Elétrica e de Computação
DCA – Departamento de Engenharia de Computação e Automação Industrial
13081-970 – Campinas – SP – Brasil

IPEP – Instituto Paulista de Ensino e Pesquisa (http://www.ipep.com.br)
Rua José de Alencar, 470
13013-040 – Campinas – SP – Brasil

Resumo

O uso de técnicas formais de teste de software tem se mostrado um bom mecanismo para revelar a presença de falhas de software. Uma das técnicas mais utilizadas é o Teste Baseado em Máquinas de Estados Finitos, uma técnica de teste funcional, suportada pela criação e uso de um modelo que descreve o comportamento do sistema sob teste. A partir deste modelo, os casos de teste podem ser gerados e executados, e os resultados dessa execução avaliados. Este artigo apresenta uma revisão desta técnica de teste, apresentando suas principais características, vantagens e desvantagens – enfocando os principais critérios de teste definidos para ela. Além disso, também são apresentadas algumas ferramentas utilizadas para a automação dos testes por meio dessa técnica e a aplicação de parte desta técnica por meio de um estudo de caso.

Palavras-chave: teste de software; teste baseado em modelos; teste baseado em MEFs.

1. Introdução

O objetivo principal da atividade de teste de software é revelar a presença de falhas no software o mais cedo possível dentro do ciclo de desenvolvimento, buscando melhorar a qualidade do produto que está sendo desenvolvido. Como a realização de um teste exaustivo é impraticável, devem ser aplicados técnicas e critérios de teste que possibilitem a seleção de um conjunto de casos de teste que possuam alta probabilidade de revelar as falhas existentes no produto, mas cuja aplicação seja viável. Também podem ser usados critérios de teste na avaliação de cobertura dos casos de teste utilizados durante a realização dos testes [19, 24].

Uma técnica de teste funcional que tem sido bastante aplicada na indústria de software é o Teste Baseado em Modelos. Os testes são realizados com base em modelos comportamentais do software, produzidos por meio de seus requisitos funcionais [1, 8]. O Teste Baseado em Modelos é uma técnica de teste bastante ampla, pois existem várias técnicas para modelar o comportamento do software e podem existir diferentes critérios de teste baseados nos modelos de cada técnica.

Entre as técnicas de modelagem, uma das mais utilizadas no Teste Baseado em Modelos são as máquinas de estados finitos (MEFs), que têm mostrado a sua excelência na atividade de modelagem, entendimento e teste de software [1, 20, 21]. Esta técnica é especialmente útil para

modelar o comportamento de sistemas reativos, os quais são essencialmente dirigidos a eventos e dominados por controle. Apesar disso, as MEFs possuem uma aplicabilidade bastante ampla e genérica, podendo ser utilizadas na modelagem de praticamente qualquer tipo de sistemas.

Devido à grande aplicabilidade das MEF, existe uma técnica específica para o Teste Baseado em Modelos chamada de Teste Baseado em Máquinas de Estados Finitos. Este artigo apresenta uma visão geral, através de uma revisão de trabalhos e autores, da técnica de Teste Baseado em MEFs. O objetivo desta revisão é apresentar aos engenheiros de software esta técnica de teste que possui grande potencial tanto para o teste manual como para o teste automatizado.

O artigo está organizado da seguinte forma: a Seção 2 apresenta as principais características das MEFs; a Seção 3 apresenta a técnica de Teste Baseado em MEFs, incluindo suas fases e alguns critérios de teste associados; a Seção 4 apresenta uma visão geral sobre a automação do Teste Baseado em MEFs e algumas ferramentas de apoio; a Seção 5 apresenta limitações encontradas nesta técnica e algumas de suas desvantagens; a Seção 6 apresenta um estudo de caso em que esta técnica foi utilizada na modelagem comportamental e no projeto de teste de um sistema; e, finalmente, a Seção 6 apresenta as conclusões e sugestões de temas de pesquisas.

2. Máquinas de Estados Finitos

Existem dois tipos de MEFs: máquinas de Mealy e máquinas de Moore. A teoria é muito similar para os dois tipos. Neste artigo são consideras principalmente as máquinas de Mealy, que modelam sistemas de estados finitos mais apropriadamente e de modo mais geral que as máquinas de Moore. Uma máquina de Mealy tem um número finito de estados e produz saídas sobre transições de estados depois do recebimento de entradas.

Uma MEF M é definida formalmente como uma 7-tupla $M = (S, s_0, I, O, D, \delta, \lambda)$, onde:

- S é um conjunto finito e não vazio de estados;
- $s_0 \in S$ é o estado inicial;
- I é um conjunto finito de entradas, que pode incluir a entrada nula;
- O é um conjunto finito de saídas, que pode incluir a saída nula;
- $D \subseteq S \times I$ é o domínio da especificação;
- $\delta: D \rightarrow S$ é a função de transições de estados;
- $\lambda: D \rightarrow O$ é a função de saída.

Uma transição de estados de M é representada por $\delta(si, i) = sj$ e a saída obtida pela realização dessa transição de estados é representada por $\lambda(si, i) = o$. Isso significa que, quando M está em um estado atual $si \in S$, ao receber uma entrada $i \in I$, ela se move para um próximo estado $sj \in S$, produzindo uma saída $o \in O$.

Dependendo da utilização da MEF, existem algumas suposições que devem ser feitas em relação às suas características. Essas suposições dizem respeito ao determinismo, a completude e aos tipos de conexão das MEFs, as quais são apresentadas a seguir.

Uma MEF M pode ser determinística ou não determinística. Se $|\delta(s, i)| = 1 \ \forall s \in S$ e $\forall i \in I$, então ela é determinística, ou seja, a partir de um estado s e sobre uma entrada i, a máquina segue apenas uma única transição até um próximo estado. Caso contrário, ela é não determinística, ou seja, a máquina pode seguir mais que uma transição e conseqüentemente produzir saídas diferentes.

Uma MEF M pode ser completamente ou incompletamente definida. Se $D = S \times I$, então a máquina é completamente definida, ou seja, a partir de um estado s e sobre uma entrada i, existe um próximo estado especificado pela função de transições de estados δ e uma saída especificada pela função de saída λ. Caso $D \subset S \times I$, então a máquina é incompletamente definida, ou seja, a partir de certos estados e sobre algumas entradas, os próximos estados e saídas não são especificados.

Uma MEF M pode ser fortemente ou fracamente conectada. Se para cada par de estados (si, sj) existe uma seqüência de entradas que leva a máquina M de si para sj, então a máquina é fortemente conectada, ou seja, a partir de qualquer estado si pode se chegar a qualquer outro estado sj. Caso contrário, a máquina é fracamente conectada, ou seja, a partir de um estado si não pode se chegar a qualquer outro estado sj.

2.1. Diagrama de Transições de Estados

Os diagramas de transições de estados, também chamados de grafos de estados, são muito úteis para a visualização das informações contidas nas MEFs. Uma representação desse tipo de diagrama é apresentada na Figura 1. Uma MEF M pode ser representada por um grafo dirigido G = (V, E), onde:

- V é um conjunto de vértices, que representa o conjunto de estados S de M. Cada v ∈ V representa um estado s ∈ S;
- E = {(vi, vj; i/o), vi, vj ∈ V} é um conjunto de arcos dirigidos, que representa as transições de M. Cada e = (vi, vj; i/o), representa uma transição de estados de vi para vj, com entrada i e saída o.

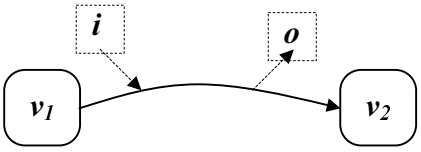

Figura 1 – Representação do Diagrama de Transições de Estados de uma MEF

2.2. Tabela de Transições de Estados

As tabelas de transições de estados são muito úteis para o armazenamento e manipulação computacional das informações contidas nas MEFs. Uma MEF M pode ser representada por uma tabela de transições de estados, conforme apresentado na Figura 2, onde cada linha representa um estado e cada coluna representa uma entrada. Para uma combinação de um estado atual si e uma entrada i, a posição correspondente da tabela especifica o próximo estado s e a saída o.

	i_1	i_2	...	i_n
s_1	s / o	s / o	...	s / o
s_2	s / o	s / o	...	s / o
...	...	...	...	...
s_n	s / o	s / o	...	s / o

Figura 2 – Tabela de Transições de Estados de uma MEF

3. Teste Baseado em MEFs

O Teste Baseado em MEFs é uma técnica de teste funcional que utiliza informações sobre o comportamento funcional do software para a realização do teste. O comportamento é descrito por meio de uma MEF, construída a partir dos requisitos funcionais do software. Esta técnica possui um grande potencial para ser utilizada no teste de grandes sistemas, por meio da automação da atividade de teste de software. Com a automação do Teste Baseado em MEFs, grande variedade e

quantidade de casos de teste podem ser gerados a partir da MEF, usando diferentes critérios. Depois de gerados, um *test driver* pode então executar os casos de teste no sistema sob teste [8, 9, 22, 23].

Essa abordagem de teste oferece uma promessa considerável na redução do custo da geração de teste, no aumento da eficiência dos testes, e na redução do ciclo de teste. O Teste Baseado em MEFs pode ser especialmente eficaz para os softwares que são alterados freqüentemente, pois uma vez que o investimento inicial foi feito para criar a MEF, pequenas alterações podem ser facilmente realizadas [9]. Além disso, dado uma mesma MEF, muitos tipos de cenários podem ser exercitados e grandes áreas da aplicação sob teste podem ser cobertas. Assim, essa técnica de teste é resistente ao "paradoxo do pesticida", em que os testes se tornam menos eficientes com o decorrer do tempo porque as falhas que eles são capazes de detectar já foram consertadas [22].

O Teste Baseado em MEFs é realizado geralmente nas fases apresentadas na Figura 3.

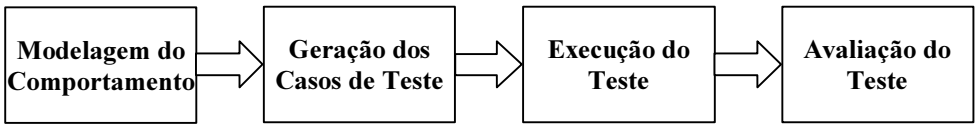

Figura 3 – Fases do Teste Baseado em MEFs

- **Modelagem do comportamento do sistema:** nessa fase, é desenvolvido um modelo que descreve o comportamento do sistema a ser testado por meio de um MEFs;
- **Geração dos casos de teste:** nessa fase, a MEF é utilizada para a geração dos casos de teste. Existem vários algoritmos que podem ser utilizados para esse propósito, os quais são descritos posteriormente;
- **Execução do teste:** nessa fase, os casos de teste são aplicados ao software sob teste;
- **Avaliação do teste:** nessa fase, os resultados obtidos durante a execução do teste são comparados com os resultados esperados.

3.1. Modelagem da MEF

O primeiro passo no Teste Baseado em MEFs é o desenvolvimento de um modelo comportamental do sistema a ser testado por meio do uso da técnica de modelagem MEFs. A modelagem é a fase mais fundamental dessa técnica de teste, já que o resto das fases (geração de casos de teste, execução e avaliação dos testes) depende da precisão do modelo criado [10].

De acordo com Robinson [19], o desenvolvimento do modelo comportamental é um processo incremental que requer que os responsáveis pela modelagem levem em consideração, simultaneamente, muitos aspectos do sistema a ser testado. Como primeira tarefa na modelagem do comportamento do sistema, os responsáveis pela modelagem devem obter uma impressão geral da funcionalidade do sistema, por meio do uso da própria aplicação e da especificação.

3.2. Geração dos Casos de Teste

Existem infinitamente mais casos de testes do que se pode realmente executar sobre o programa. Portanto, com o objetivo de se distinguir os casos de teste que possuem uma maior probabilidade de detecção da presença de falhas de que os outros, certos critérios de teste devem ser estabelecidos. Existem vários critérios que podem ser usados para desenvolver os casos de teste a partir de uma MEF. A escolha de quais critérios utilizar também é uma decisão muito importante, pois a eficácia da atividade da atividade de teste depende dos critérios utilizados [10].

Para o Teste Baseado em MEFs, os casos de teste não representam a abordagem simples em que apenas um valor de entrada (ou um conjunto de valores) deve ser aplicado ao programa sob

teste e apenas um valor (ou um conjunto de valores) é esperado como saída. Ao invés disso, os casos de teste devem ser descritos em função dos elementos existentes no próprio modelo. Portanto, para modelos descritos por meio de MEFs, os casos de teste devem ser descritos em função dos estados e das transições de estado. Assim, os dados de entrada de um caso de teste podem ser descritos como uma seqüência de ações que levam a uma seqüência de estados, e a saída esperada do caso de teste pode ser descrita como as saídas esperadas para cada transição e um estado final.

As MEFs podem ser consideradas essencialmente grafos; assim, muitos algoritmos da teoria de grafos são usados como base de critérios de teste. A maioria desses algoritmos está relacionada ao conceito de caminho. Um caminho é uma seqüência de transições através dos elementos do modelo (estados) e que pode ser usado para definir um cenário de uso real do sistema. A diferença dos critérios entre si, é que cada um deles possui uma forma diferente de selecionar os caminhos a serem testados e gerar os casos de teste que exercitem esses caminhos. A seguir encontra-se uma breve descrição de alguns desses critérios, os quais são bastante utilizados por testadores de empresas produtoras de software.

- **Seleção Aleatória:** essa é uma das escolhas mais populares para esse propósito, que permite que os caminhos sejam selecionados aleatoriamente, pegando qualquer transição disponível a partir de um estado. A natureza aleatória dessa escolha significa que elas tendem a produzir combinações não usais que testadores humanos não se incomodariam em testar. Entretanto, esse critério tende a ser muito ineficiente para cobrir um grafo muito grande, por ser infinito [18, 19, 21].
- **Todos os Estados:** esse é um dos critérios mais simples existentes, que define que todos os estados dentro da MEF devem ser executados. Similarmente ao critério "Todos os Nós" do teste estrutural, esse critério possui uma cobertura muito pequena da MEF [19].
- **Todas as Transições:** esse também é um dos critérios mais simples existentes, sendo um pouco mais exigente que o critério "Todos os Estados". Esse critério define que todas as transições dentro da MEF devem ser executadas. Similarmente ao critério "Todos os Arcos" do teste estrutural, esse critério também possui uma cobertura pequena da MEF [17, 19].
- **Caminho Mais Curto Primeiro:** esse critério inicia a seleção de caminhos por meio do estado inicial e incrementalmente escolhe todos os caminhos de tamanhos 2, 3, 4, etc. Essa é essencialmente uma busca em profundidade, a qual não oferece nenhuma garantia de otimização, mas os caminhos são geralmente eficientes [19, 20].
- **Carteiro Chinês:** esse critério, baseado no algoritmo do carteiro chinês, gera um caminho a partir da MEF que exercita todas as transições na MEF com o menor número de passos possível. Por meio desse critério, consegue-se a menor seqüência de testes que fornecerá cobertura completa da MEF inteira [18, 20, 21] esse critério são os critérios Carteiro Chinês Escolhendo Estados e Carteiro Chinês Capacitado [20, 21].
- **Caminho Mais Provável Primeiro:** esse critério é usado para grafos cujas transições possuem uma determinada probabilidade de ocorrer (correntes de Markov). Esse critério define que os caminhos com maior probabilidade são executados primeiro. Esse critério pode ser indicado para teste de funcionalidade extensiva, teste de regressão e para a verificação de que uma versão do software tem as funcionalidades básicas [19, 20].
- **Todas as Combinações:** esse é o critério que possui maior cobertura, já que todas as combinações de transições dentro da MEF devem ser executadas. Entretanto, similarmente ao critério "Todos os Caminhos" do teste estrutural, dependendo do tamanho da MEF, esse critério pode ser considerado inviável. No teste de software baseado em estados, essa abordagem é também chamada de *switch cover* [3, 17, 18].

Além desses critérios, existem critérios que foram definidos formalmente com base em trabalhos de pesquisa realizados especificamente para o Teste Baseado em MEFs. De acordo com

[12], os mais conhecidos desses critérios formais são: Transition Tour [16]; método W [6]; método Wp [12]; método Distinguishing Sequence – DS [13]; e método Unique-Input-Output – UIO [23].

Embora a utilização desses critérios ofereça uma maior garantia de que o software possui alta qualidade e alta confiabilidade por meio da realização dos testes, eles são critérios mais complexos, apresentando conseqüentemente um maior custo em sua aplicação. Por isso, esses critérios não são muito utilizados nas empresas produtoras de software. As definições desses critérios, por serem geralmente bastante complexas, não são pertinentes a este artigo.

Os casos de teste derivados por cada um desses critérios detectam a presença de qualquer tipo de falha de saída da implementação. Ou seja, se a implementação segue a MEF, que modela o seu comportamento, exceto pela saída produzida por certas transições de estado, a presença da falha será detectada durante a execução dos casos de teste. Entretanto, a presença de falhas de transferência – isso é, falhas no alcance do próximo estado por uma transição – não são sempre detectadas. Apesar disso, considerando que o número de estados da implementação permanece dentro de um certo limite, os métodos W e DS detectam a presença de todos as falhas da implementação – tanto as falhas de saída como as falhas de transição.

Os critérios escolhidos para a seleção de casos de teste podem ser usados também como critérios de adequação de casos de teste. Ou seja, eles podem ser utilizados para verificar qual parte do modelo já foi coberta por meio da execução de um determinado conjunto de casos de teste.

A geração de um conjunto de casos de teste – por meio do modelo comportamental do software e um critério de seleção de casos de teste – não é uma tarefa trivial. Assim, deve-se levar em consideração a possibilidade de automação, principalmente, dessa fase do Teste Baseado em MEFs. Para isso, deve haver um algoritmo que, baseado no modelo construído na fase anterior, gere casos de teste que satisfaça o critério escolhido. Esse tipo de informação é geralmente armazenado em scripts que são criados automaticamente e servem de entrada para a próxima fase – a fase de execução do teste.

3.3. Execução do Teste

A execução do teste consiste basicamente em executar os casos de teste gerados na fase anterior. Essa fase envolve o descobrimento de como simular as ações do usuário, a partir dos casos de testes gerados, de modo que o software aja como se estivesse em seu ambiente pretendido. Essa tarefa também pode ser feita manualmente, ou então, automatizada.

A automação dessa fase do Teste Baseado em MEFs pode ser realizada por um *test driver* (chamado também de *test harness*), que é um programa que aplica uma seqüência de entradas – os casos de teste – ao sistema sob teste. O *test driver* deve obter uma seqüência de transições a partir do critério escolhido, ler a seqüência de teste, entrada por entrada, determinar qual transição deve ser aplicada para cada entrada, e fazer com que a aplicação execute aquela transição, levando a um novo estado.

3.4. Avaliação do Teste

A fase de avaliação do teste envolve a verificação do resultado da execução do teste contra algum tipo de especificação. Assim, nessa fase, os resultados obtidos durante a execução do teste, os quais são tipicamente ou as saídas de transições ou o estado final, são comparados com os resultados esperados. Normalmente, essa fase de avaliação do teste é realizada pelo mesmo módulo responsável pela execução dos casos de teste, isso é, o *test driver*.

Fundamental a realização dessa fase é a existência de algum tipo de oráculo. Um oráculo é um mecanismo que verifica se a aplicação comportou-se corretamente ou não. Assim, o oráculo é uma entidade independente que determina se o resultado observado no software depois que o teste foi

executado satisfaz as expectativas, isso é, se as saídas corretas foram produzidas, ou se as seqüências de controle corretas foram seguidas [10].

O desenvolvimento de um oráculo não é uma tarefa trivial e pode ser tão complexo como o desenvolvimento da própria aplicação sob teste, representando um dos maiores custos da atividade de teste. Entretanto, um dos grandes benefícios do Teste Baseado em MEFs é a habilidade de se utilizar o modelo comportamental do sistema para a verificação das saídas obtidas [17]. Assim, é possível saber, a partir do modelo, quais transições deveriam estar disponíveis a partir de cada estado e a saída esperada de cada transição, podendo assim saber qual o estado final esperado [18].

4. Automação do Teste Baseado em MEFs

Uma importante característica do Teste Baseado em MEFs é que ele possibilita a automação completa da atividade de teste de software. Essa automação é possível devido ao suporte oferecido pelo modelo comportamental da aplicação criado por meio de MEFs, a partir do qual os casos de teste podem ser automaticamente gerados e executados, e os resultados avaliados para se obter os resultados do teste [8, 19, 20].

A automação oferece a essa técnica uma grande agilidade à execução de testes. Dado um modelo comportamental do software sob teste e um *test driver*, grandes quantidades de casos de teste podem ser geradas e executadas. Além disso, se o comportamento do software for modificado, o seu modelo comportamental pode ser facilmente atualizado e novamente serem gerados novos conjuntos inteiros de casos de teste [8, 9].

Robinson [19, 20] mostra que, além de agilidade, a automação oferece também grande flexibilidade à execução de testes. Por exemplo, a geração dos casos de teste pode ser direcionada para diferentes áreas de foco de teste, tais como teste de stress, teste de regressão e a verificação de que uma versão do software possui algumas funcionalidades básicas. Além disso, entradas que levam a falhas conhecidos podem ser temporariamente desabilitadas do modelo. Assim, quaisquer casos de teste baseados nesse novo modelo evitarão as falhas conhecidas e nenhuma única linha do código de teste precisa ser mudada para isso.

A automação das diversas fases do Teste Baseado em MEFs pode ser realizada, em geral, por dois tipos diferentes de componentes. Uma forma é utilizar ferramentas genéricas para qualquer sistema a ser testado. A outra forma é utilizar módulos que devem ser criados para cada sistema a ser testado. Pode-se ainda, utilizar os dois tipos de componentes descritos, sendo cada tipo para uma fase distinta da técnica de teste.

4.1. Ferramentas para o Teste Baseado em MEFs

Uma das ferramentas mais citadas na literatura é a TestMaster (produzida pela Teradyne, Inc.) [2, 7]. Esta ferramenta utiliza a tecnologia de referência de modelo (MRT – Model Reference Technology) para fornecer a geração de teste automática a partir de um modelo de MEF da aplicação sob teste. TestMaster compreende três componentes principais: uma ferramenta de edição gráfica, um gerador de programas de teste e um depurador de modelos. TestMaster pode ser usada para se criar um modelo detalhado e a partir desse modelo gerar rapidamente uma série de casos de teste. O número de casos de teste gerados é determinado pela quantidade de cobertura de teste requerida. A saída de TestMaster para cada caso de teste, quando executado o test driver, é capaz de configurar o equipamento de teste, executar uma série de testes e desenvolver relatórios.

Uma outra ferramenta é a Visual Test (produzida pela Rational Software) [18, 19, 20]. Esta ferramenta, utilizada por engenheiros de teste da Microsoft, é na realidade uma linguagem de teste que pode ser usada para automatizar todas as fases do Teste Baseado em MEFs. Ela pode ser usada para: criar uma lista de transições de estados a partir de um modelo de estados finitos; criar

seqüências de ações para serem executadas; criar *scripts* que executam as seqüências criadas; e ainda criar oráculos para determinar se a aplicação comportou-se corretamente durante a execução das seqüências de ações ou não.

Uma outra ferramenta que pode ser utilizada no apoio do Teste Baseado em MEFs é a ferramenta POKE-TOOL (Potential Uses Criteria Tool for Program Testing), uma ferramenta de apoio ao teste desenvolvida na Faculdade de Engenharia Elétrica e de Computação – FEEC – da Universidade Estadual de Campinas – Unicamp [4, 5, 15]. Esta ferramenta foi originalmente desenvolvida para propiciar a aplicação de critérios d teste estrutural principalmente como critérios de adequação. Entretanto, ela foi estendida para também dar suporte a aplicação de critérios de teste funcional baseados em MEFs Estendidas [11].

No que se refere ao uso dos critérios de teste funcional, a POKE-TOOL é utilizada para verificar o quanto um conjunto de casos de teste, executado sobre um programa, satisfaz um certo critério de teste em relação a sua especificação em MEFs Estendidas. Dado um conjunto de casos de teste, a ferramenta pode ser utilizada para realizar uma análise de cobertura, que consiste em determinar o percentual de elementos requeridos da MEF Estendida, por um determinado critério, que foram exercitados pelo conjunto de casos de teste.

Além disso, a ferramenta POKE-TOOL também pode ser utilizada como auxílio na seleção de casos de teste. Dado um conjunto de casos de teste e um critério de teste funcional, a análise de cobertura permite determinar quais os elementos requeridos da MEF Estendida que ainda não foram exercitados, fornecendo subsídios para a seleção de novos casos de teste que aumentem o grau de cobertura. A ferramenta oferece suporte a um amplo conjunto de critérios de teste funcional baseados em MEFs Estendidas. Entretanto, alguns deles podem ser usados em MEFs tradicionais, os quais são: Todos os Estados; Todos as Transições; e Todos os Caminhos.

5. Limitações e Desvantagens

Embora as MEFs sejam uma excelente ferramenta para modelagem, entendimento e teste, existem algumas limitações em relação aos mecanismos de modelagem oferecidos por ela. Uma destas limitações é a não existência de uma hierarquia nesta técnica de modelagem. Assim, o modelo pode se tornar complexo e de difícil manutenção para sistemas grandes. Para solucionar esse problema, existem algumas extensões dessa técnica que suportam modelos hierárquicos, nos quais um estado pode ser substituído por uma chamada a um outro modelo que define o comportamento dentro do estado. Modelos hierárquicos permitem que comportamentos complexos sejam decompostos em modelos de mais baixo nível.

Uma outra limitação é que elas possibilitam apenas a modelagem do fluxo de controle do software, não possibilitando a modelagem de seu fluxo de dados. Assim, os critérios de teste baseados em MEFs existentes não consideram informações associadas ao fluxo de dados do software [11]. Além da falta de hierarquia e da modelagem de fluxo de dados, existem outras limitações das MEFs tradicionais que prejudicam o seu uso como técnica de modelagem para o teste de software. Exemplos dessas limitações são a falta de mecanismos para a modelagem de concorrência, comunicação, processos distribuídos e paralelismo.

Poucas desvantagens do Teste Baseado em MEFs são apresentadas na literatura. Uma delas é que uma quantidade de esforço é necessária para desenvolver o modelo inicial para sistemas relativamente complexos. Outra desvantagem é que um *test driver* deve ser desenvolvido para cada aplicação sob teste, já que isso requer que os testadores sejam capazes de programar, como no caso do uso da ferramenta Visual Test [19, 20]. Além disso, ainda existe o problema de que muitas vezes os requisitos do *software* não estão completos, tornando difícil a geração dos modelos que traduzem a realidade do *software* implementado e que deve ser testado.

6. Estudo de Caso: TeleMicro - Software de Apoio a Ligações Telefônicas

Esta seção apresenta a aplicação do Teste Baseado em MEFs na geração de casos de teste para o sistema TeleMicro – um sistema que visa oferecer apoio à realização de ligações telefônicas efetuadas por meio de um computador, possibilitando a seus usuários um melhor gerenciamento e acompanhamento de suas ligações. Os requisitos deste sistema são agrupados em funcionalidades de manutenção de cadastro, realização de consultar e controle do sistema telefônico.

As funcionalidades de controle telefônico são: Efetuar Ligação, Receber Ligação e Programar Ligação Automática. Para este estudo de caso, apenas a funcionalidade Receber Ligação foi considerada. Esta funcionalidade é usada quando alguma ligação de um *chamador* deve ser recebida pelo sistema. A ligação pode ser recebida pelo próprio usuário ou automaticamente pelo sistema caso o usuário não a atenda. A Figura 3 apresenta a MEF modelada para esta funcionalidade.

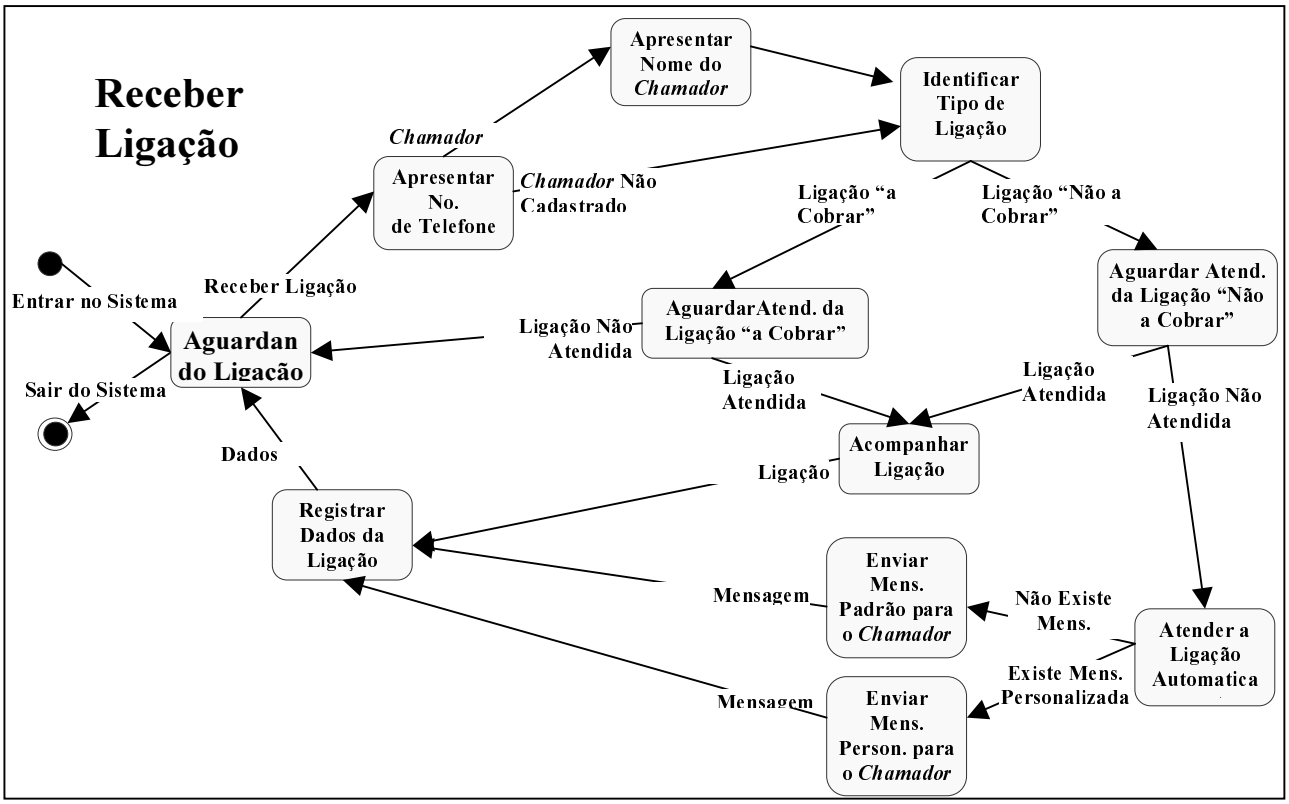

Figura 3 – MEF da Funcionalidade Receber Ligação do Sistema TeleMicro

O sistema recebe uma ligação, apresenta o número de telefone ao usuário e, se cadastrado, o nome do *chamador*. Depois o tipo de ligação é identificado. Se a ligação for atendida, o sistema apresenta o tempo decorrido e o provável valor da ligação durante a ligação. Se uma ligação "não a cobrar" não for atendida, o sistema envia uma mensagem gravada, que pode ser geral ou personalizada. Por último, o sistema armazena as informações da ligação recebida no banco de dados para posterior análise e retorna para o estado de aguardando o recebimento de uma ligação.

Para esta especificação em MEFs foi gerado um conjunto de casos de teste que tem o objetivo de satisfazer o critério de teste Todas as Transições. A geração dos casos de teste foi realizada com o apoio da ferramenta POKE-TOOL, descrita na seção 3 deste artigo. A ferramenta POKE-TOOL fornece os arcos primitivos da MEF. Estes arcos são um subconjunto de todos os arcos que sendo exercitados é garantido que todos os demais arcos também o serão. O conjunto de arcos primitivos da MEF da Figura 4 é apresentado na Tabela 1.

Tabela 1 – Arcos Primitivos da MEF da Funcionalidade Receber Ligação

No.	Estado Inicial	Estado Final
1	Apresentar No. de Telefone do *Chamador*	Apresentar Nome do *Chamador*
2	Apresentar No. de Telefone do *Chamador*	Identificar Tipo de Ligação
3	Aguardar Atend. da Ligação "a Cobrar"	Aguardando Ligação
4	Aguardar Atend. da Ligação "a Cobrar"	Acompanhar Ligação
5	Aguardar Atend. da Ligação "Não a Cobrar"	Acompanhar Ligação
6	Atender a Ligação Automaticamente	Enviar Mens. Padrão para o *Chamador*
7	Atender a Ligação Automaticamente	Enviar Mens. Person. para o *Chamador*
8	Aguardando Ligação	Estado Final

Através de uma análise dos dados da Tabela 1, é possível perceber que bastam 4 casos de teste para exercitar todos os 8 arcos primitivos da MEF modelada. Um exemplo de um dos casos de teste (o que exercita os arcos primitivos 2, 6 e 8) é apresentado na Tabela 2. Este caso de teste é descrito em alto nível, sem apresentar os detalhes de verificação de resultados.

Tabela 2 – Exemplo de Caso de Teste

Caminho a ser exercitado: Estado Inicial; Aguardando Ligação; Apresentar No. de Telefone do *Chamador*; Identificar Tipo de Ligação; Aguardar Atend. da Ligação "Não a Cobrar"; Atender a Ligação Automaticamente; Enviar Mens. Padrão para o *Chamador*; Registrar Dados da Ligação; Aguardando Ligação; Estado Final.		
Dados de Entrada		**Resultado Esperado**
Estado Inicial	**Entrada/Ação**	**Próximo Estado**
Estado Inicial	Entrar no Sistema	Aguardando Ligação
Aguardando Ligação	Receber Ligação	Apresentar No. de Telefone do *Chamador*
Apresentar No. de Telefone do *Chamador*	*Chamador* Não Cadastrado	Identificar Tipo de Ligação
Identificar Tipo de Ligação	Ligação "Não a Cobrar"	Aguardar Atend. da Ligação "Não a Cobrar"
Aguardar Atend. da Ligação "Não a Cobrar"	Ligação Atendida Automaticamente	Atender a Ligação Automaticamente
Atender a Ligação Automaticamente	Não Existe Mensagem Personalizada	Enviar Mens. Padrão para o *Chamador*
Enviar Mens. Padrão para o *Chamador*	Mensagem Enviada	Registrar Dados da Ligação
Registrar Dados da Ligação	Dados Registrados	Aguardando Ligação
Aguardando Ligação	Sair do Sistema	Estado Final
Condições para a execução do caso de teste: fazer com que o sistema receba (de forma simulada) uma ligação "Não a Cobrar" de um número de telefone que não tem um contato associado cadastrado na agenda do usuário. A ligação não deve ser atendida. Depois que a ligação for atendida automaticamente, a mensagem enviada e os dados registrados na base de dados, o usuário deve escolher sair do sistema.		

7. Conclusão

Este artigo apresentou o Teste Baseado em Máquinas de Estados Finitos, que é uma técnica de teste funcional que se apresenta como uma forma eficiente e adaptável de teste de software. Essa técnica de teste é suportada pela criação e uso de um modelo, elaborado por meio de MEFs, que descreve o comportamento do sistema sob teste, a partir do qual os casos de teste podem ser gerados e executados, e os resultados dessa execução avaliados.

O Teste Baseado em MEFs é realizado geralmente em quatro fases: 1) modelagem do comportamento do sistema por meio de MEFs; 2) geração dos casos de teste; 3) execução do teste; e 3) avaliação do teste. A mais fundamental dessas fases é a modelagem do comportamento do sistema, pois é da precisão do modelo criado que depende o resto das fases. A fase de geração dos casos de teste também é bastante importante, pois a eficácia da atividade de teste vai depender dos critérios de seleção de testes utilizados.

Esta técnica de teste possui um grande potencial para ser utilizada no teste de grandes sistemas, por meio da automação completa da atividade de teste de software. Por meio dessa

automação, grandes números de casos de teste podem ser gerados a partir de um modelo comportamental do *software*, usando diferentes critérios. Depois de gerados, um *test driver* pode então executar os casos de teste no sistema sob teste. Assim, muitas áreas de foco de teste podem ser implementadas e diferentes níveis de cobertura do modelo podem ser atingidos por meio do uso do mesmo modelo e do mesmo *test driver*.

Uma grande parte da literatura tem se dedicado a apresentar estudos de casos relacionados à aplicação da abordagem de Teste baseado em MEFs em vários tipos de softwares [1, 7, 8, 9, 14]. Em todos eles, as execuções dos testes apresentaram ótimos resultados na detecção da presença de falhas, as quais não foram ou dificilmente seriam detectadas por abordagens de teste tradicionais. Neste artigo, também é apresentado um estudo de caso realizado, em que uma funcionalidade de um sistema foi primeiramente modelada por meio de MEFs, a qual foi usada na geração de casos de teste que devem satisfazer o critério de teste Todas as Transições.

Entretanto, para tornar essa abordagem uma solução prática, fatores econômicos devem ser levados em consideração. Uma nova técnica, para ser adotada e utilizada, deve permitir a economia de dinheiro. As métricas tradicionais usadas para justificar a compra de *softwares* ou para garantir uma mudança em um processo já existente incluem: baixo custo, aumento da qualidade e redução do tempo para o mercado. O Teste baseado em MEFs é uma técnica de teste que tem sua habilidade provada em fornecer melhorias dramáticas nos três campos dessas métricas. Assim, o maior desafio para uma aceitação mais ampla dessa técnica é a educação [1].

Como trabalhos futuros nesta área, dois principais aspectos devem ser investigados:

- Extensão dos critérios de teste funcional baseados em MEFs para técnica de modelagem Diagrama de Atividades da UML (Unified Modelling Language).
- Extensão do modelo de MEFs para representar atributos de criticidade associados à execução de estados, para que critérios de teste possam ser desenvolvidos baseados nestes atributos.

A principal evolução deste trabalho será a aplicação desta técnica de teste de software, por meio de um estudo de caso, no sistema de faturamento (*billing system*) desenvolvido pelo CPqD. Este é um sistema bastante complexo, caracterizado por registrar o uso – por cliente – de todos os serviços oferecidos por uma empresa de telecomunicações, resultando na contabilização desses serviços por meio da emissão de faturas para os clientes.

Agradecimentos

Ao CPqD Telecom & IT Solutions que, por meio do FUNTTEL – Fundo para o Desenvolvimento Tecnológico das Telecomunicações, está investindo na realização desta pesquisa, a fim de utilizar a técnica aqui descrita em seu processo de desenvolvimento de software.

Referências Bibliográficas

[1] Apfelbaum, L. e doyle, J. *Model-Based Testing*. Proceedings of The 10th International Software Quality Week, 27-30 May, 1997, San Francisco, California, USA.

[2] Berger, B.; Abuelbassal, M. E Hossain, M. *Model Driven Testing*. DNA Enterprisises, Inc., Disponível em: www.model-based-testing.com, Março de 1997.

[3] Bourhfir, C.; Dssouli, R.; Aboulhamid, E. M. *Automatic Test Generation for EFSM-Based Systems*. Publication departamentale #1043, 1996, (Disponível em www.umontreal.Ca/Labs/ Teleinfo/Publistindex.Html).

[4] Bueno, P. M. S., Chaim, M. L., Maldonado, J. C., Jino, M. e Vilela, P. R. S. *Manual do Usuário Da POKE-TOOL*. Relatório Técnico, DCA/FEEC, UNICAMP, Campinas, SP, 1995.

[5] Chaim, M. L. *POKE-TOOL – Uma Ferramenta para Suporte ao Teste Estrutural de Programas Baseado em Análise de Fluxo de Dados*. Dissertação de Mestrado, DCA/FEEC, UNICAMP, Campinas, SP, Abril de 1991.

[6] Chow, T. S. *Testing Software Design Modeled by Finite-State Machines*. IEEE Transactions On *Software* Engineering, Se(4(3)), 1978.

[7] Clarke, J. M. Automated Test Generation from a Behavioral Model. Anais da Software Quality Week Conference, Maio de 1998.

[8] Dalal, S. R.; Jain, A.; Karunanithi, N.; Leaton, J. M. e Lott, C. M. *Model-Based Testing of a Highly Programmable System*. Proceedings of The Ninth International Symposium on Software Reliability Engineering – ISSRE'98, pp. 174-178, IEEE Computer Society Press, November , 1998.

[9] Dalal, S. R.; Jain, A.; Karunanithi, N.; Leaton, J. M.; Lott, C. M.; Patton, G. C. e Horowitx, B. M. *Model-Based Testing in Practice*. Proceedings of The 21st International Conference on Software Engineering – ICSE'99, 16-22 May, 1999, Los Angeles, California, USA.

[10] El-Far, I. K. I. *Automated Construction of Software Behavior Models*. Tese de Mestrado, Florida Institure of Technology, Melbourne, Florida, Disponível em: www.model-based-testing.com, Maio de 1999.

[11] Fantinato, M. *Critérios de Teste Funcional Baseados em Máquinas de Estados Finitos Estendidas.* Submetido a SBES´2002 – Simpósio Brasileiro de Engenharia de Software, 2002.

[12] Fujiwara, S.; Bochmann, G. V.; Khendek, F.; Amalou, M. e Ghedamsi, A. *Test Selection Based on Finite State Models*, IEEE Transaction on Software Engineering, SE17(6), Junho, 1991.

[13] Gonenc, G. *A Method for the Design of Fault-Detection Experiments,* IEEE Transaction on Computer, Vol C-19, pp 551-558, Junho, 1970.

[14] Gronau, I.; Hartman, A.; Kirshin, A.; Nagin, K. E Olvovsky, S. *A Methodology and Architecture for Automated Software Testing*. IBM Research Laboratory in Haifa Technical Report, Disponível em: www.model-based-testing.com, 2000.

[15] Maldonado, J. C. *Critérios Potenciais – Usos: Uma Contribuição ao Teste Estrutural de Software.* Teste de Doutorado, DCA/FEEC, UNICAMP, Campinas, SP, Julho de 1991.

[16] Naito, S. e Tsunoyama, M. *Fault Detection for Sequenctial Machines by Transition-Tours,* Proceedings of FTCs – Fault Tolerant Computer Systems, pp. 238-243, 1981.

[17] Offutt, A. J.; Liu, S. e Abdurazik, A. *Generating Test Data from State-Based Specifications*. The Journal of Software Testing, Verification and Reliability, John Wiley & Sons, 2000.

[18] Robinson, H. *Graph Theory Techniques in Model-Based Testing*. Proceedings of The 16th International Conference on Testing Computer Software, June, 1999, Washington, USA.

[19] Robinson, H. e Rosaria, S. *Applying Models in Your Testing Process*. Information and Software Technology, Vol. 42, No. 12, pp. 815-824, Elsevier Science, September 2000.

[20] Robinson, H. *Intelligent Test Automation*. Software Testing & Quality Engineering Magazine, pp. 24-32, Software Quality Engineering, September/October, 2000.

[21] Rocha, A. R. C.; Maldonado, J. C. e Weber, K. C. *Qualidade de Software – Teoria e Prática*. Prentice Hall, 2001.

[22] Sabião, S. B.; Martins, E. e Ambrosio, A. M. *ConData: A Tool for Automation Specification-Based Test Case Generation for Communication Systems*. Proceedings of the 33rd Hawaii International Conference on System Sciences, 04-07 January, 2000, Maui, Hawaii, USA.

[23] Sabnani, K. K. e Dahbura, A. T. *A Protocol Testing Procedure*, Computer Networks and ISDN System, Vol. 15, N. 4, pp. 285-297, 1988.

[24] Sato, F., Munemori, J., E Mizuno, T. *Test Sequence Generatin Method Based on Finite Automata – Single Transition Checking Method Using W Set*, (em japonês), Trans. EIC, vol. J72-B-I, no. 3, pp. 183-192, 1989.