Geração de Testes de Interoperabilidade e Robustez para uma Aplicação Espacial

Anderson C. Weller [1], Eliane Martins [2], Fátima Mattiello-Francisco [3]

[1] Instituto Federal do Espírito Santo (Ifes) – Campus Santa Teresa
Santa Teresa – ES – Brasil

[2] Instituto de Computação – Universidade Estadual de Campinas (Unicamp)
Av. Albert Einstein 1251 – 13081-970 – Campinas – SP – Brasil

[3] Instituto Nacional de Pesquisas Espaciais (INPE)
São José dos Campos – SP – Brasil

andersoncw@ifes.edu.br, eliane@ic.unicamp.br, fatima.mattiello@inpe.br

Abstract. *This paper presents an approach for interoperability and robustness testing of distributed real-time embedded systems (DRE), search-based on UML state machine model, and with the use of test purposes with the objective to avoid the state space explosion problem for test case generation. The model represents the interaction between two components that interact via a faulty communication channel. We applied the approach to a space application service that was used in a previous study. Results show that the approach is as useful as the previous one to cover the test purposes, but with the differential of the faulty environment is also modeled and to the use of a search-based test case generation technique.*

Resumo. *Apresentamos uma abordagem para teste de interoperabilidade e robustez de sistemas distribuídos de tempo real (DRE), baseada em busca em modelo de máquina de estado UML e no uso de propósitos de teste com o objetivo de evitar o problema de explosão do espaço de estado na geração de casos de teste. O modelo representa a interação entre dois componentes que interagem através de um canal de comunicação com defeito. Aplicamos a abordagem em um serviço de aplicativo espacial que foi usado em um estudo anterior. Os resultados mostram que a abordagem é tão útil quanto a anterior para cobrir os objetivos do teste, mas com o diferencial do ambiente defeituoso ser modelado e ao uso de técnica de geração de casos de teste baseada em busca.*

1. Introdução

Atualmente, o uso de sistemas embarcados é cada vez mais popular, à medida que se tornam uma parte essencial de muitos sistemas técnicos complexos, que vão desde cuidados com a saúde, automotivo, telefones, aeroespacial, apenas para citar alguns. Devido aos avanços na tecnologia de *hardware*, os sistemas embarcados estão sendo cada vez mais compostos por muitos processadores interconectados via redes, formando sistemas embarcados distribuídos em tempo real (DREs). Essa tendência também é verdadeira para os sistemas de satélite, geralmente construídos em torno de um

barramento do sistema, cujo o controlador está conectado aos vários subsistemas, incluindo as cargas úteis (que implementam os objetivos da missão). Com o aumento da complexidade dos DREs, o *software* se tornou predominante nesses sistemas, e devido a necessidade de interação com o ambiente do mundo real, algumas características devem ser consideradas durante o teste [Bringmann and Krüamer 2008]:

- DREs demandam mais qualidades, como confiabilidade ou segurança, do que sistemas convencionais, pois nem sempre é possível uma reinicialização após uma falha. E, como a interação com o mundo físico não é totalmente previsível, não bastam testes funcionais, sendo importante determinar a sua robustez.
- Sistemas embarcados geralmente residem em plataformas dedicadas, o que requer o co-design de *software* e *hardware*. Os testes são executados em diferentes plataformas durante os vários níveis de integração, portanto, os casos de teste devem ser portáveis para diferentes plataformas;
- Como um sistema distribuído, um DRE precisa de testes que avaliem o ambiente de comunicação e o gerenciamento das conexões entre as máquinas;
- DREs geralmente precisam satisfazer restrições em tempo real na interação com o mundo real. Como consequência, o teste deve considerar não apenas se o sistema produz respostas corretas, mas também se ele tem o comportamento adequado na ocorrência de violações de restrições de tempo.

A derivação manual de casos de teste é comum na prática, mas deve ser reduzida ao mínimo, pois isso torna a atividade de teste dispendiosa, desestruturada e, portanto, difícil de reproduzir, além de estar sujeita a erros e omissões. O teste baseado em modelo (MBT), que usa modelos abstratos para gerar casos de teste, é uma abordagem viável para o teste de DREs pois visa a geração automática de casos de teste abstratos (independentes de plataforma). O MBT apresenta outras vantagens, mas também apresenta alguns desafios, em especial, ao considerar o teste de robustez de sistemas complexos de DREs, cujos componentes devem interagir de acordo com as especificações, mesmo na presença de falhas. Um desses desafios é o problema de escalabilidade associado aos métodos de teste clássicos que usam técnicas exaustivas para a geração de casos de teste. Esses métodos envolvem a geração de casos de teste a partir de uma especificação completa, exigindo uma pesquisa de casos de teste em um enorme espaço de estado [Salva and Laurençot 2007]. Esse problema é crítico para testes funcionais e piora ao considerar aspectos de robustez e tempo real para componentes de comunicação.

Em um trabalho anterior, propusemos o InRob (*INteroperability and ROBustness testing*) [Mattiello-Francisco 2009], para verificar a interoperabilidade e robustez relacionadas às restrições de tempo dos sistemas embarcados em tempo real (RTES). O InRob é uma abordagem de geração de caso de teste baseada em modelo na qual os dois componentes de comunicação são modelados. O InRob foi instanciado usando um TIOA (*Timed Input Output Automata*) para representar o comportamento dos componentes de comunicação na presença de falhas de tempo. Um modelo aumentado é obtido de um modelo funcional, introduzindo os desvios das restrições de tempo e a robustez esperada associada a cada desvio adicionado. Um emulador de falha (FEM) foi considerado como parte da arquitetura de teste, responsável pela injeção das falhas de tempo durante o tempo de execução. Uma limitação dessa abordagem é que ela lida com atrasos e tempos limites, mas não considera outras falhas de comunicação,

como corrupção de dados, por exemplo. Outro ponto é que o FEM, embora faça parte da arquitetura, não é explicitamente modelado, o que dificulta a reutilização de seu design para outros sistemas similares.

As contribuições deste artigo são: primeiro, neste artigo, apresentamos uma extensão para o InRob, designada como InRob-UML, e sua aplicação em um sistema real. O método é centrado em notações UML para representar aspectos estáticos e dinâmicos da arquitetura de teste, sendo que o comportamento dos componentes em interação é modelado com um subconjunto de máquinas de estado UML. O FEM é explicitamente modelado, permitindo a representação de um modelo de falha mais rico. Para evitar um problema de escalabilidade, é usada uma abordagem baseada em propósito de teste para geração de casos de teste, como foi o caso do InRob, mas uma abordagem baseada em busca é usada para orientar a busca pelos casos de teste que cubram os propósitos de teste. Ele pode gerar casos de teste que abrangem parte de controle e dados de um modelo de estado UML. Dessa forma, não é preciso determinar a priori o tamanho do caso teste, pois o algoritmo de busca, além da cobertura, procura por casos de teste com o menor tamanho possível.

O método foi aplicado a três estudos de caso. O primeiro foi um sistema de controle de passagem de nível (*Generalized Railroad Crossing* – GRC), muito utilizado na literatura para representar a modelagem de sistemas de tempo real [Leveson and Stolzy 1985], publicado em trabalho anterior [Weller et al. 2015]. O método também foi aplicado nos testes de um sistema de um nano-satélite, desenvolvido em colaboração com o INPE (Instituto Nacional de Pesquisas Espaciais) e com a UFSM (Universidade Federal de Santa Maria) [Batista et al. 2018]. Este trabalho apresenta a aplicação do método na modelagem e geração de casos de teste para um subsistema de aquisição de dados científicos da plataforma espacial do projeto MIRAX (Monitoramento e Imageamento de uma região rica em fontes de Raios X) [Mattiello-Francisco 2009].

O restante do artigo está estruturado da seguinte forma. Na Seção 2 é apresentada a fundamentação teórica. Na Seção 3 é apresentado o método proposto. Na Seção 4, as ferramentas utilizadas para a aplicação do método. Na Seção 5 é descrito o estudo empírico para avaliação da abordagem. A Seção 6 mostra alguns trabalhos relacionados. E, a Seção 7 conclui o artigo, dando instruções para trabalhos futuros.

2. Fundamentação Teórica

O objetivo do teste de interoperabilidade é determinar se duas ou mais implementações se comunicam corretamente e se elas se comportam conforme o esperado por suas respectivas especificações durante esta comunicação [Desmoulin and Viho 2009]. Geralmente, os testes consideram um contexto de interoperabilidade um para um, no qual o sistema em teste compreende dois componentes em teste (CUT).

O teste de robustez, por outro lado, tem por objetivo validar se um sistema tem um comportamento aceitável na presença de entradas inesperadas ou inválidas. Os casos de teste no teste de robustez são caracterizados por *workload* e *faultload*. O primeiro contém as entradas que são submetidas ao sistema em teste (SUT), enquanto o *faultload* introduz falhas no SUT. Neste estudo, estamos mais preocupados com falhas de comunicação entre os CUTs, que incluem corrupção, duplicação, perda ou atraso de mensagens [Natella et al. 2016]. Como as falhas são introduzidas durante o tempo de

execução, um aspecto importante é quando acionar falhas, que podem depender do tempo ou do evento [Yano et al. 2011]. No primeiro caso, a injeção ocorre após um tempo (aleatório ou especificado pelo usuário), enquanto no segundo caso, a injeção é ativada quando um evento específico ocorre durante a execução, como, por exemplo, uma mensagem específica é transmitida por um dos CUT.

Para aumentar a controlabilidade durante o teste, escolhemos a injeção acionada por evento. A combinação de testes de interoperabilidade e robustez apresenta vários desafios para os testes baseados em modelo. O modelo é complexo, pois deve representar os componentes que interagem e o comportamento de cada componente. Essa complexidade implica em um espaço de estado enorme e, portanto, a pesquisa de casos de teste não pode ser exaustiva. O problema se torna ainda pior ao introduzir aspectos relativos ao comportamento na presença de falhas. A seguir apresentamos uma proposta que visa mitigar esse problema.

3. Apresentação do método proposto

O método InRob-UML é uma extensão do InRob, com algumas diferenças, resumidas na Tabela 3.1.

Tabela 3.1. Diferenças entre os dois métodos

	InRob	InRob-UML
Artefatos	- perfil do serviço - modelo nominal do serviço (TIOA) - perigos de tempo - modelo aumentado do serviço (TIOA) -propósito de teste	- perfil do serviço - modelo do serviço (diagrama de classes + diag. de estados) - modelo de falhas (temporais e de comunicação) - modelo do ambiente com falhas (diagr. de classes + diag. de estados) - propósitos de teste (diagrama de sequência)
Passos	- modelagem do serviço - geração dos casos de teste - execução dos casos de teste	- modelagem do serviço - modelagem do ambiente com falhas - validação do modelo - geração dos casos de teste abstratos - concretização dos casos de teste - execução dos testes e análises dos resultados

Assim como o InRob, o método é composto de três fases distintas. Na Fase A é feita a priorização e a modelagem dos serviços; na Fase B são obtidos os casos de teste abstratos e na Fase C os casos de teste são concretizados e executados. Neste trabalho o foco são as fases A e B. A Figura 3.1 apresenta as fases A e B do método; as etapas destacadas pela cor cinza diferem do método InRob.

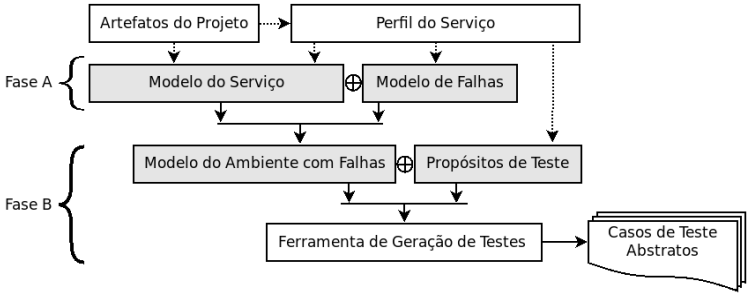


Figura 3.1. Fases A e B do método InRob-UML.

Os Artefatos do Projeto são as informações que detalham o projeto e descrevem o funcionamento esperado dos subsistemas. Junto com a definição do Perfil de Serviço, servem como base para a especificação do Modelo de Serviço, que é composto por diagramas de classe e seus respectivos diagramas de estados.

Deve-se agrupar em Perfis de Serviço as funcionalidades do sistema que tenham características comuns, identificar a importância de cada um deles (através de cálculos estatísticos [Mattiello-Francisco 2009]) e priorizar a ordem de geração dos casos de teste abstratos. O Perfil de Serviço irá nortear a definição do Modelo de Serviço, do Modelo de Falhas e dos Propósitos de Teste (ou TP, do inglês *Test Purpose*).

Para validar a robustez do *software* seleciona-se alguns parâmetros (Modelo de Falhas) que representem eventos não previstos na especificação, como falhas na comunicação ou problemas de desempenho dos módulos, que representem algum perigo ao funcionamento do sistema. Com base no Modelo de Falhas é montado o diagrama de estados do mecanismo emulador de defeitos (FEM), que especifica o comportamento e ações do canal de comunicação durante a transmissão de mensagens entre os subsistemas em teste. Esse diagrama é integrado ao Modelo de Serviço para gerar o Modelo do Ambiente com Falhas, do qual serão derivados os casos de teste abstratos.

Os Propósitos de Teste (TP) são partes da especificação que se deseja testar, e pode ser visto como um cenário parcial de execução. Para defini-lo, utilizamos diagramas de sequência UML com os cenários que são relevantes para a interação, definidos no Perfil de Serviço escolhido. A partir desse diagrama, deve-se selecionar uma mensagem alvo que o caso de teste deve abranger e, opcionalmente, outras mensagens intermediárias pelos quais o caso de teste também deve disparar.

4. Ferramentas utilizadas na aplicação do método

O método não preconiza o uso de nenhuma ferramenta específica para a geração de casos de teste, mas deve-se atentar aos critérios de seleção disponíveis na ferramenta escolhida, para evitar o problema da explosão combinatória de casos de teste.

Neste trabalho foi utilizada ferramenta StateMutest [Cardoso 2015], desenvolvida pela Universidade Estadual de Campinas em parceria com membros da Universidade Federal de São Carlos (UFSCar) e do Instituto Nacional de Pesquisas Espaciais (INPE). Com ela é possível, entre outras funcionalidades, criar modelos de máquina de estados, gerar casos de teste para eles e aplicar os casos de teste gerados contra um modelo. Na versão utilizada, a StateMutest utiliza um modelo executável que é produzido pelo SMC (*State Machine Compiler*) [Rapp 2015].

O SMC permite criar um modelo de estados executável, sendo que ele é um subconjunto do modelo de estados da UML, sendo que cada um deles fica contido em um mapa. Assim como as regiões nos modelos da UML, os mapas contêm estados, transições, ações e guardas, assim como na UML. Existem também diretivas *push* e *pop* que permitem a transição de estados entre esses mapas, permitindo montar máquinas comunicantes entre si, ou sub-máquinas. Essa comunicação entre os mapas cria uma espécie de "pilha de execução", onde o topo fica o mapa em execução e nos níveis inferiores ficam os mapas que dispararam uma transição *push*, aguardando os respectivos retornos, simulando uma troca de mensagens síncronas entre os modelos.

A StateMutest faz uso de uma abordagem baseada em busca para a geração de casos de teste, ou seja, utiliza algoritmos de otimização para buscar soluções que atendam a um objetivo (algoritmos mono-objetivo) ou mais objetivos (algoritmos multi-objetivos). No caso, é utilizada MOST (*Multi-Objective Search-based Testing*

approach from EFSM) [Yano et al. 2011], que utiliza um algoritmo multi-objetivo para a geração de casos de teste a partir de modelos de estado. Ela se baseia em propósitos de teste para guiar a busca por soluções que atendam a dois objetivos: (1) cobrir as transições que constam do propósito de testes e (2) procurar por casos de teste mais curtos que atendam a (1). As soluções geradas representam um balanço entre as funções objetivo, com o intuito de encontrar uma sequência de entrada de tamanho mínimo, porém longa o suficiente para cobrir uma transição alvo e as transições de dependência (ou de cobertura) escolhidas pelo usuário e que fazem parte do propósito de teste.

5. Aplicação ao estudo de caso

O projeto MIRAX (Monitor e Imageador de Raios X), desenvolvido pela Divisão de Astrofísica do Instituto Nacional de Pesquisas Espaciais (INPE), envolveu uma missão de satélite científico para monitoramento de uma região rica em fontes de Raios X.

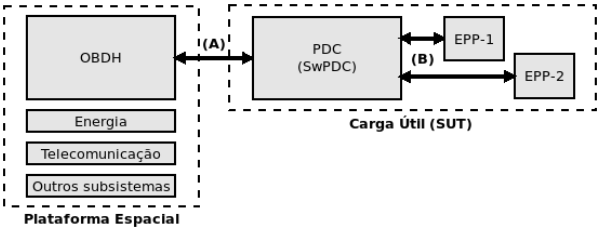


Figura 5.1. Componentes do sistema MIRAX [Mattiello-Francisco 2009].

Ele é composto por uma Plataforma Espacial (PE) e a sua Carga Útil (SUT), ver Figura 5.1. A PE é formada por subsistemas de controle da missão, e um deles é o OBDH (*On Board Data Handling*), que tem a função de requisitar dados dos subsistemas de coleta de dados científicos, que ficam como carga útil do satélite. Dentro da carga útil encontram-se câmeras EPP (*Event Packet Processor*), para aquisição de dados, e o computador PDC (*Payload Data Computer*) com o seu *software* embarcado (SwPDC). O SwPDC solicita ao subsistema EPP a captura e o envio das imagens, depois as armazena para posterior envio ao OBDH sempre que requisitado.

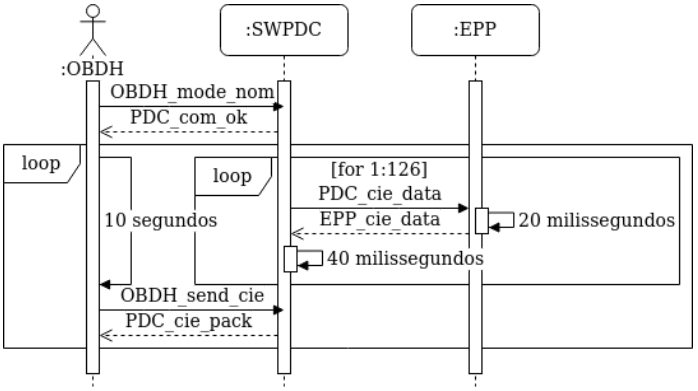


Figura 5.2. Serviço de dados científicos - exemplo de cenário.

Seguindo o InRob-UML, avaliamos a comunicação entre dois subsistemas ao utilizarem um canal de comunicação falho. Assim, conforme o perfil de serviço de aquisição de dados científicos, selecionado em [Mattiello-Francisco 2009], iremos modelar o SUT em uma arquitetura Mestre/Escravo, englobando os subsistemas SwPDC e EPP, para gerar os casos de teste abstratos de interoperabilidade e robustez dessa interação. Conforme Figura 5.1, a interface (A) será utilizada para envio dos

comandos de teste e para a obtenção dos resultados retornados pelo SUT, e a interface (B) será manipulada pelo FEM para direcionar o caminho dos testes realizados.

Conforme o diagrama de sequência UML da Figura 5.2, o serviço inicia com um comando “mode_nom”, enviado pelo OBDH, colocando o SUT em modo nominal de operação, e sua confirmação é feita através de uma mensagem “com_ok”. Após entrar em modo nominal, o PDC passa a enviar solicitações de dados científicos para o EPP a cada 40ms, e aguarda o retorno dos dados em até 20ms. Caso esse tempo seja extrapolado, o PDC envia uma solicitação de retransmissão. Em paralelo, o OBDH requisita dados científicos ao PDC a cada 10s. Na Tabela 5.1 é apresentado um conjunto de requisitos do subsistema SwPDC no projeto MIRAX.

Tabela 5.1. Requisitos de tempo selecionados para os testes

Código	Requisitos do SwPDC
1	O PDC envia uma solicitação de dados para o EPP a cada 40 milissegundos.
2	O PDC armazena os dados científicos retornados pelo EPP.
3	O EPP deve responder ao PDC em até 20 milissegundos após o envio da requisição.
4	Se o EPP não responder à requisição do PDC em até 20 milissegundos, o PDC envia uma solicitação de retransmissão.
5	O OBDH requisita dados científicos para o PDC a cada 10 segundos.
6	O PDC envia os dados científicos para OBDH.
7	O serviço de aquisição de dados científicos deve ser um serviço cíclico, com garantia de execução de 95% do tempo.

5.1. Modelagem estática

Primeiro, modelamos a estrutura estática do SUT através de um diagrama de classes, contendo os elementos utilizados na arquitetura de testes (ver Figura 5.3). Nele, temos o PDC que está ligado ao EPP através de um canal de comunicação, que é substituído pelo Emulador de Falhas no canal (FEM) durante os testes de interoperabilidade e robustez.

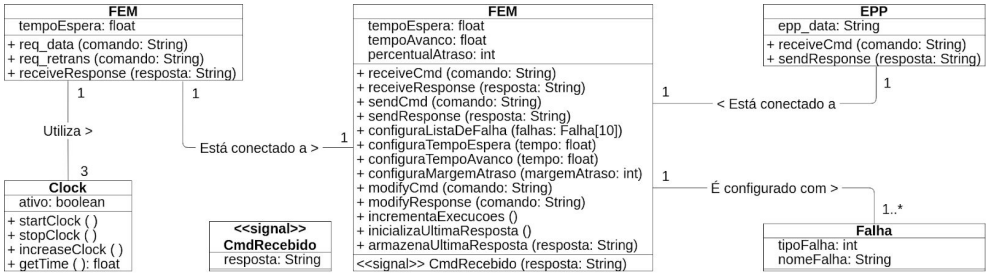


Figura 5.3. Diagrama de classes com os componentes do MIRAX.

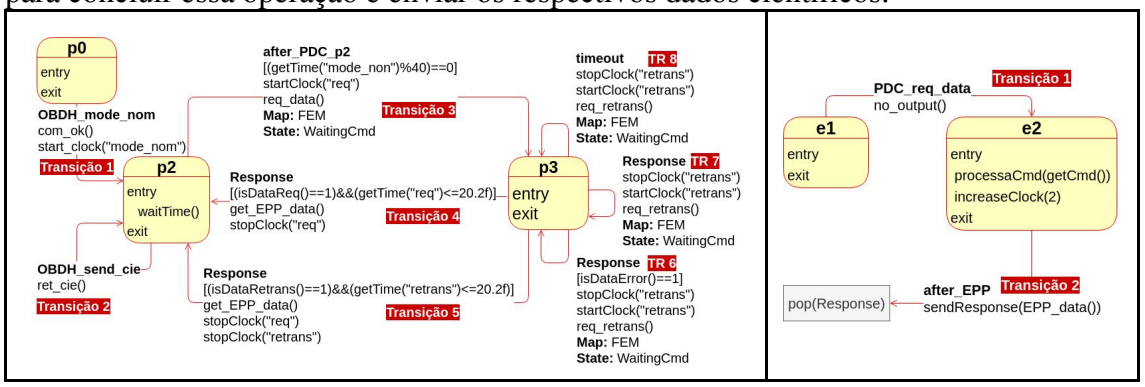
No PDC temos dois métodos diferentes para envio de comandos para o EPP: “req_data” e “req_retrans”. Além disso, o PDC trabalha com três *clocks* diferentes durante a execução: um para simular o comportamento periódico do OBDH, outro para controlar as requisições feitas pelo PDC e mais um outro para controlar o tempo decorrido para os pedidos de retransmissão feitos ao EPP. O FEM intermedia a comunicação entre o PDC e o EPP através dos métodos “receiveCmd”, “receiveResponse”, “sendCmd” e “sendResponse”.

5.2. Modelagem dinâmica

O comportamento do PDC está descrito no diagrama de estados da Figura 5.4 (a). O PDC inicia no estado “p0”, e muda para “p2” assim que receber de OBDH uma

mensagem colocando-o em modo nominal. A partir desse momento, o PDC passa a solicitar dados científicos ao EPP a cada 40ms. Estando no estado “p3”, o PDC aguarda os dados vindos do PDC dentro de um prazo máximo de 20ms, e solicita retransmissão caso esse tempo tenha excedido (*Timeout*) ou tenha ocorrido algum problema com a recepção dos dados (*Response*[*isDataError*]).

O comportamento do EPP está representado no diagrama de estados na Figura 5.4 (b). Ao receber uma solicitação do PDC, o EPP passa do estado inicial “e1” para “e2”, que inicia o processamento dessa requisição, e tem um tempo máximo de 2ms para concluir essa operação e enviar os respectivos dados científicos.



(a) Modelo de estados do PDC (b) Modelo de estados do EPP

Figura 5.4. Modelos de comportamento.

A ligação entre esses dois subsistemas é feito através de um canal de comunicação, que no InRob-UML é representado pelo FEM. Seu comportamento é apresentado na Figura 5.5, sendo equivalente ao apresentado em [Weller et al. 2015]. Ao ser disparada a transição 3 do PDC, *after*(40ms), o controle de execução do modelo é repassado para o estado *WaitingCmd* do mapa do FEM. Assim que o FEM interceptar a mensagem (por *CmdReceived*), através de uma condição de guarda (ver transições 1, 2, 3 ou 5 da Figura 5.5) ele exerce uma ação diferente nessa transmissão, podendo injetar uma falha ou um atraso nessa comunicação. Cumpre notar que essas transições direcionam o controle de execução para o mapa do EPP e o FEM fica aguardando o retorno de uma mensagem *Response* (ver as transições 10, 11 e 12 da Figura 5.5).

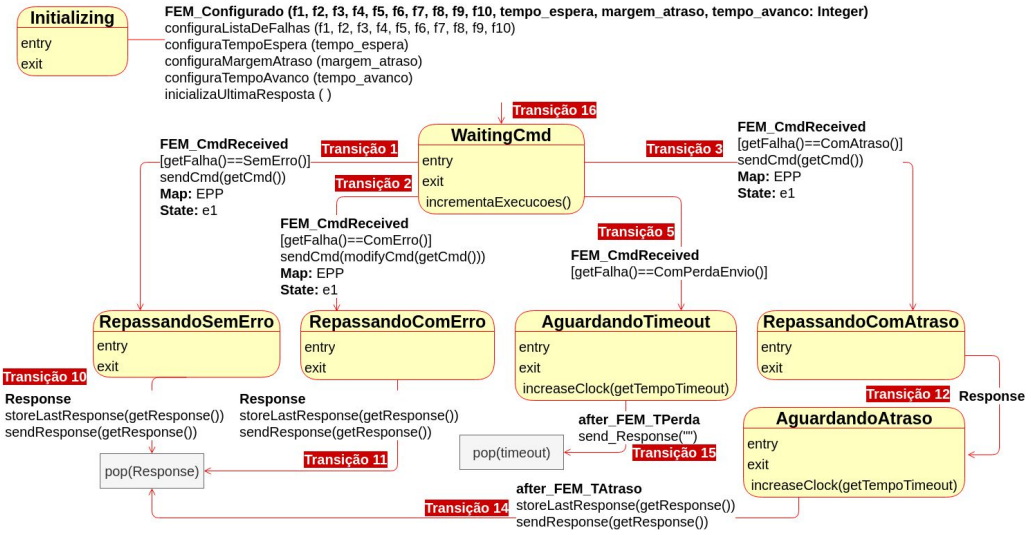


Figura 5.5. Modelo de comportamento do FEM.

Para a validação dos modelos foi utilizada a própria StateMutest para a animação, uma vez que permite a criação de um modelo executável via SMC. Foram criados alguns casos de teste manualmente, e após a execução de cada um deles, conferiu-se visualmente na ferramenta se foram disparadas as transições esperadas e atingidos os estados previstos durante a criação do caso de teste.

5.3. Propósitos de Teste

Após as devidas correções no modelo executável decorrentes dos resultados da animação do modelo, partimos para a geração automática dos casos de teste com base em alguns cenários. Esses cenários serviram de base para a definição dos TPs, através da seleção da transição alvo (t-alvo) e das transições de cobertura (t-cob), que servirão de entrada para a StateMutest. Os cenários escolhidos estão resumidos na Tabela 5.2. SEQ-falhas indica a lista de falhas previamente selecionadas para o FEM (um número indica uma falha específica, dois números separados por um traço indica a faixa de falhas que o algoritmo pode escolher durante a busca dos casos de teste, T-atraso define o tempo de *Timeout*, T-margem informa o percentual aceitável de atraso das mensagens (relativo ao tempo de *Timeout*).

Tabela 5.2. Cenários utilizados para os testes

TP ID	Descrição	t-alvo	t-cob	SEQ-falhas	T-atraso (ms)	T-margem (%)
TP01	Cenário Normal: Recebe dados EPP	PDC 4	Vazio	{1-9;1-9;1-9;...}	20	1
TP02	Cenário Robustez: *Timeout* + Recebimento	PDC 5	PDC 7 ou 8	{1-9;1-9;1-9;...}	20	1
TP03	Cenário Robustez: Corrompido + Recebimento	PDC 5	PDC 6	{1-9;1-9;1-9;...}	20	1
TP04	Perda Mensagem Pedido	EPP 2	FEM 5	{1-9;1-9;1-9;...}	20	1
TP05	Perda Mensagem Resposta	PDC 5	FEM 17	{1-9;1-9;1-9;...}	20	1
TP06	Perda Pedido Retransmissão	PDC 8	FEM 5, 17 e 20	{1-9;1-9;1-9;...}	20	1
TP07	Perda Mensagem Resposta (Com falhas fixas)	PDC 5	FEM 17	{4;1;1;1;1;1;...}	20	1

5.4. Objetivos dos experimentos

Para avaliar o uso da Inrob-UML em termos da geração de casos de teste, realizamos experimentos com o objetivo de responder às seguintes questões:

1. Os casos de teste cobrem os propósitos de teste (TPs) dos cenários selecionados?
2. Há alguma característica nos TPs que facilite a busca dos casos de teste?
3. É possível gerar casos de teste que direcionem a atuação de um FEM para a avaliação da interoperabilidade e robustez de dois subsistemas?
4. É possível representar todas as informações estáticas e dinâmicas de subsistemas interoperantes utilizando apenas o formalismo definido no padrão UML?

Como métricas a serem utilizadas para auxiliar a avaliação de cada uma dessas questões, temos:

1. A efetividade dos casos de teste obtidos, em termos de cobertura dos TPs.
2. Número de avaliações feitas pelo algoritmo na seleção de casos de teste.
3. O número de casos de teste únicos obtidos ao definir manualmente a sequência de falhas aplicadas pelo FEM, e ao deixar que essa sequência seja escolhida pelo algoritmo de busca.
4. A quantidade de adaptações necessárias ao padrão dos Diagramas de Estado UML para que seja possível modelar a interoperabilidade entre os subsistemas.

5.5. Resultados obtidos com a aplicação da InRob-UML

Os casos de testes foram gerados utilizando a ferramenta StateMutest apresentada na Seção 4. Para levar em conta a aleatoriedade da busca evolutiva, executou-se 50 vezes o algoritmo de otimização para cada um dos TPs.

A Tabela 5.3 resume os resultados obtidos durante os experimentos. Nele temos o número total de casos de teste obtidos pelas execuções do algoritmo. Desses casos de teste, verificamos manualmente se as transições são disparadas na mesma ordem em que aparecem no propósito de testes, ou seja, a transição alvo é disparada após a transição de cobertura. Os casos de teste que não atendiam a essa característica foram considerados como **inválidos**. Dado que o algoritmo de geração de testes é executado várias vezes para um mesmo propósito de teste, podemos obter casos de teste válidos que passem pelas mesmas transições antes de chegar à transição alvo. Neste caso, consideramos que esses resultados, apesar de válidos, são casos de teste **repetidos**, mesmo que eles tenham tamanhos diferentes, pois o que consideramos foi somente o cenário escolhido. Desconsiderando esses casos repetidos, chegamos aos casos **únicos**.

Tabela 5.3. Sumário dos resultados obtidos

TP ID	Número de Execuções	Nº Casos de Teste	Casos Válidos	Casos Válidos por Execução	Casos Repetidos	Casos Únicos	Casos Únicos por Execução
TP01	50	54	54	108%	46	8	16%
TP02	50	96	95	190%	38	57	114%
TP03	50	50	46	92%	27	19	38%
TP04	50	50	27	54%	17	10	20%
TP05	50	50	46	92%	27	19	38%
TP06	50	38	35	70%	0	35	70%
TP07	50	50	50	100%	46	4	8%
Média	50	55,43	50,43	100,9%	28,71	21,72	43,43%

Devido às características da ferramenta utilizada, normalmente são gerados apenas um caso de teste, por execução, que atenda a todas as transições do TP, assim como nos TPs de 3 a 6. Porém, durante a execução do experimento TP01 houve o retorno de 4 soluções extras que atendiam ao propósito de teste. Já no TP02, como definimos duas transições de cobertura alternativas para o propósito de teste, bastava que uma delas estivesse no caso de teste para que este fosse considerado válido. Assim, obtivemos praticamente dois casos de teste por execução do algoritmo para o TP02.

Avaliando os resultados obtidos em relação à questão 1, sobre a eficácia em termos de cobertura dos TPs, obtivemos, em média, mais casos de teste do que o número de execuções do algoritmo. Porém, se não considerarmos o TP02, onde obtivemos cerca de dois casos de teste por execução, então chegamos a 86% de sucesso na obtenção de casos de teste válidos por execução do algoritmo.

Em relação à questão 2, sobre a existência de parâmetros que auxiliem a busca dos casos de teste, não foi possível encontrar evidências que justifiquem esta afirmação, visto que o algoritmo despendeu praticamente o mesmo esforço em cada um dos TPs, conforme pode ser visto na última coluna da Tabela 5.4. Porém, caso o propósito de teste permita transições de cobertura alternativos, assim como no TP02, então podemos ter um número maior de soluções a cada execução do algoritmo.

Tabela 5.4. Dados da execução da ferramenta de geração de casos de testes.

TP ID	Comprimento do caso de teste				Número de avaliações por execução			
	Média	Máximo	Mínimo	Desvio Padrão	Média	Máximo	Mínimo	Desvio Padrão
TP01	8,4	15	6	2,0	106022,7	199124	2066	61861,4
TP02	19,3	31	11	4,8	101489,6	200258	168	58842,5
TP03	19,1	33	13	4,7	96478,0	197570	592	53107,3
TP04	12,1	17	9	1,9	96365,2	186798	4318	55475,6
TP05	19,6	29	14	3,5	94999,4	199098	1102	58224,8
TP06	34,3	54	24	7,5	96084,4	197462	5112	62133,9
TP07	15,7	21	14	1,6	88056,2	191162	6	60018,9

Em relação à questão 3, quanto a geração de casos de teste que direcionem as ações do FEM, observamos que é possível obter um maior número de casos únicos de execução quando deixamos para o algoritmo de busca a escolha das ações do FEM, conforme pode ser comparado entre os resultados entre TP05 e TP07 na Tabela 5.3. Pela Tabela 5.4, verificamos que o número de avaliações desses dois TPs são bem próximos, não justificando gasto de tempo na escolha manual das ações do FEM.

E, por último, sobre a representação de todas as características do modelo UML na ferramenta utilizada (questão 4), precisamos realizar algumas adaptações ao modelo executável, visto que ele foi construído a partir da linguagem SMC. Uma delas é quanto à comunicação entre os diagramas de estados, feita através de transições *push* e *pop*, o que não permite a execução em paralelo. Além disso, transições de tempo after precisam ter identificações únicas para facilitar a implementação dos casos de teste. Assim, é possível identificar o tempo de espera correto em cada uma dessas transições.

5.6. Discussão e lições aprendidas

Os experimentos realizados nos permitiram avaliar as características do método InRob-UML na modelagem de um sistema embarcado de tempo-real para testes de interoperabilidade entre dois subsistemas, combinando com os testes de robustez em que se considera os dois subsistemas são interligados por um canal de comunicação falho, que é emulado pelo FEM. Como o FEM representa o comportamento de um canal de comunicação, ele independe das implementações em teste, e pode ser utilizado em qualquer sistema que tenha a arquitetura Mestre/Escravo. Tanto é que foi utilizado em três estudos de caso, requerendo poucos ajustes no modelo.

De acordo com as ferramentas e algoritmos utilizados obtivemos, para 50 execuções do experimento, uma taxa de sucesso de 100% na obtenção de casos de teste válidos, e de 43% de casos válidos e únicos, evidenciando que a solução escolhida não retorna sempre os mesmos resultados.

Uma limitação da versão da ferramenta StateMutest utilizada neste estudo foi o uso do SMC para a criação do modelo executável, sendo necessárias algumas adaptações ao padrão UML para que os diagramas de estados pudessem ser interligados pelo diagrama do FEM, como as transições *after* e a utilização de transições *push* e *pop*. Outra adaptação feita é quanto ao tratamento do *Timeout* gerado pelo FEM, como *push* e *pop* emulam uma comunicação síncrona, é necessário incluir condições de guarda para avaliar se a mensagem retornada pelo *pop* é um *Timeout* ou uma resposta entregue com atraso.

A realização dos experimentos possuem algumas limitações quanto às ameaças à validade de construção, interna, externa e de conclusão. Em relação às validades de construção, a elaboração dos experimentos poderia ter sido influenciado pelos resultados esperados. Para evitar isso, escolhemos os TPs de acordo com os perfis de serviço selecionados em outros trabalhos.

Quanto às ameaças à validade interna, elas podem vir pela forma como o experimento foi realizado. Assim, para reduzir a probabilidade de falhas, cada teste foi repetido 50 vezes sob as mesmas condições, mas com diferentes inicializações e com o apoio de um algoritmo randomizado.

As ameaças quanto à validade externa vêm do fato que, apesar do número de experimentos, avaliamos apenas alguns cenários em duas aplicações do método proposto. Assim, mesmo que esses exemplos representem as características de interoperabilidade de sistemas embarcados de tempo real, não há garantia de que foram avaliadas todas a especificidades dos demais sistemas dessa área.

Outro ponto a ser destacado é quanto a característica da ferramenta de busca que define os TPs através de transições alvo, mas não tem opção para definir o número de vezes que essa transição deva ser disparada para se alcançar o objetivo de teste, como por exemplo, passar 3 vezes pela a transição alvo. Assim, mesmo que seja possível obter casos de teste que alcancem a transição alvo mais do que uma vez, a ferramenta normalmente retorna resultados que passem apenas uma vez pela transição alvo, visto que é utilizado um algoritmo de otimização.

Já em relação à validade de conclusão, poderíamos ter resultados diferentes caso o número aplicações do método e de cenários avaliados fosse maior, principalmente quanto a possíveis adaptações do diagrama de estados, por conta das características da linguagem SMC utilizada na construção do modelo executável.

6. Trabalhos relacionados

Entre as abordagens de teste de robustez baseadas em modelo propostas, algumas aumentam o modelo do sistema com o modelo de falha considerado para teste de robustez [Saad-Khorchef et al. 2007, Batth et al. 2007]; outras criam modelos separados para representar o comportamento do sistema na presença de falhas [Ambrosio et al. 2005]. O uso de aspectos também é proposto para representar o comportamento robusto, permitindo a reutilização do modelo de robustez em aplicações similares [Ali et al. 2012]. Essas abordagens estão relacionadas ao teste de conformidade e não consideram a representação de um canal de comunicação com defeito.

No que diz respeito à geração de casos de teste com restrições de tempo, o trabalho [Salva and Laurençot 2007] também considera uma abordagem baseada em propósito de teste para testar a conformidade e a robustez, abordando o comportamento temporal, usa o TIOA como modelo de teste, e trata apenas de uma implementação. Outra abordagem usa autômatos temporizados e sistemas de transição simbólica para lidar com aspectos de dados e temporização de um sistema em tempo real [Andrade et al. 2011], mas seu foco também está em testes de conformidade. O InRob [Mattiello-Francisco 2009], por outro lado, é uma abordagem de teste de interoperabilidade e robustez para testar DREs. Seu foco também estava nas restrições

de tempo, mas o modelo de falha de robustez foi representado como um modelo aumentado dos componentes em interação. Como modelo de teste, o InRob usou o TIOA para representar o comportamento dos componentes e também é uma abordagem baseada em finalidade de teste. Em um estudo anterior [Weller et al. 2015], avaliamos a abordagem com um estudo de caso da literatura, mas neste trabalho usamos aplicativos reais de DRE e focamos nos aspectos de geração de casos de teste.

7. Conclusões e Trabalhos Futuros

Este trabalho apresenta o método InRob-UML de geração de testes baseados em modelos, para avaliação da robustez e da interoperabilidade entre subsistemas embarcados de tempo real, ele é um extensão do InRob, baseada na modelagem UML e com a inclusão do comportamento do canal de comunicação (FEM) como uma entidade do modelo. A modelagem se baseia em notações da UML, como o diagrama de classes e o diagrama de estados, este último sendo a base para a geração dos casos de teste. O InRob-UML define um modelo de falhas para representar os eventos não previstos na especificação, como problemas de desempenho nos módulos e a inserção de falhas pelo canal de comunicação, que possam vir a afetar o funcionamento dos subsistemas, ou a interação entre eles.

Como uma técnica de geração de testes baseados em modelos, este método permite que o desenvolvedor gere casos de teste, mesmo antes da implementação dos subsistemas. Além disso, ao escolher o UML, que é amplamente difundido no meio acadêmico e comercial, pode-se aproveitar os próprios diagramas de classes, de estados e de sequência, produzidos durante o projeto do sistema, como base para a geração dos casos de teste e para a seleção dos propósitos de teste.

A escolha da ferramenta StateMutest para a criação do modelo executável permite, durante a geração dos casos de teste, a escolha automática dos parâmetros de tempo e das sequências de falhas a serem aplicadas durante os testes. Assim, de acordo com o propósito de teste, estes parâmetros e falhas também podem ser predeterminados, o que possibilita uma maior flexibilidade na busca dos casos de teste. A geração de casos de teste que utiliza algoritmos baseados em busca para a seleção com base em propósitos de teste, que visa não somente a cobertura destes propósitos, mas também gerar casos de teste com o menor tamanho possível.

Como trabalho futuro, a InRob-UML será utilizada em outros sistemas embarcados. Quanto à ferramenta StateMutest, uma nova versão está em projeto para superar as limitações da versão atual.

Referências

Ali, S., Briand, L. C., and Hemmati, H. (2012). Modeling robustness behavior using aspect-oriented modeling to support robustness testing of industrial systems. Software & Systems Modeling, 11(4):633–670.

Ambrosio, A. M., Martins, E., Vijaykumar, N. L., and de Carvalho, S. V. (2005). Systematic generation of test and fault cases for space application validation. In DASIA 2005-Data Systems in Aerospace, volume 602.

Andrade, W. L., Machado, P. D., Jéron, T., and Marchand, H. (2011). Abstracting time and data for conformance testing of real-time systems. In Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on, pages 9–17. IEEE.

Batista, C. L. G., Weller, A. C., Martins, E., and Mattiello-Francisco, F. (2018). Towards increasing nanosatellite subsystem robustness. Acta Astronautica.

Batth, S. S., Vieira, E. R., Cavalli, A., and Uyar, M. Ü. (2007). Specification of timed EFSM fault models in SDL. In Formal Techniques for Networked and Distributed Systems–FORTE 2007, pages 50–65. Springer.

Bringmann, E. and Krämer, A. (2008). Model-based testing of automotive systems. In Software Testing, Verification, and Validation, 2008 1st International Conference on pages 485–493. IEEE.

Cardoso, W. F. F. (2015). StateMutest: uma ferramenta de apoio ao teste baseado em modelos de estado estendidos. Master's thesis, Instituto de Computação, Unicamp, Campinas, SP, Brazil.

Desmoulin, A. and Viho, C. (2009). Formalizing interoperability for test case generation purpose. International journal on software tools for technology transfer, 11(3):261–267.

Leveson, N. G. and Stolzy, J. L. (1985). Analyzing safety and fault tolerance using time petri nets. In International Joint Conference on Theory and Practice of Software Development, pages 339–355. Springer.

Mattiello-Francisco, M. F. (2009). InRob – Uma abordagem para testes de interoperabilidade e de robustez de subsistemas de tempo-real intensivos em software. Doutorado em engenharia eletrônica e computação, Instituto de Tecnológico de Aeronáutica, São José dos Campos, SP.

Natella, R., Cotroneo, D., and Madeira, H. S. (2016). Assessing dependability with software fault injection: A survey. ACM Computing Surveys (CSUR), 48(3):44.

Rapp, C. W. (2015). SMC: The State Machine Compiler. http://smc.sourceforge.net. Accessed on Abr. 10, 2020.

Saad-Khorchef, F., Rollet, A., and Castanet, R. (2007). A framework and a tool for robustness testing of communicating software. In Proceedings of the 2007 ACM symposium on Applied computing, pages 1461–1466. ACM.

Salva, S. and Laurençot, P. (2007). Generation of tests for real-time systems with test purposes. RTNS'07, page 35.

Weller, A. C., Martins, E., and Mattiello-Francisco, F. (2015). InRob-UML: uma Abordagem para Testes de Interoperabilidade e Robustez baseados em Modelos. In 9th Brazilian Workshop on Systematic and Automated Software Testing – SAST 2015, pages 71–80.

Yano, T., Martins, E., and de Sousa, F. (2011). MOST: A Multi-objective Search-Based Testing from EFSM. In Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on, pages 164–173.